(19)日本国特許庁(JP)　(12)公表特許公報(A)　(11)特許出願公表番号
特表2004-533029
(P2004-533029A)
(43)公表日　平成16年10月28日(2004.10.28)

(51)Int.Cl.[7]	F I		テーマコード（参考）
G06F 12/02	G06F 12/02	510A	5B025
G06F 12/00	G06F 12/02	570A	5B060
G11C 16/02	G06F 12/00	597U	
	G11C 17/00	612F	
	G11C 17/00	601E	

審査請求 未請求　予備審査請求 有　（全 93 頁）

(21)出願番号　特願2002-558275(P2002-558275)
(86)(22)出願日　平成14年1月7日(2002.1.7)
(85)翻訳文提出日　平成15年7月18日(2003.7.18)
(86)国際出願番号　PCT/US2002/000366
(87)国際公開番号　WO2002/058074
(87)国際公開日　平成14年7月25日(2002.7.25)
(31)優先権主張番号　09/766,436
(32)優先日　平成13年1月19日(2001.1.19)
(33)優先権主張国　米国(US)

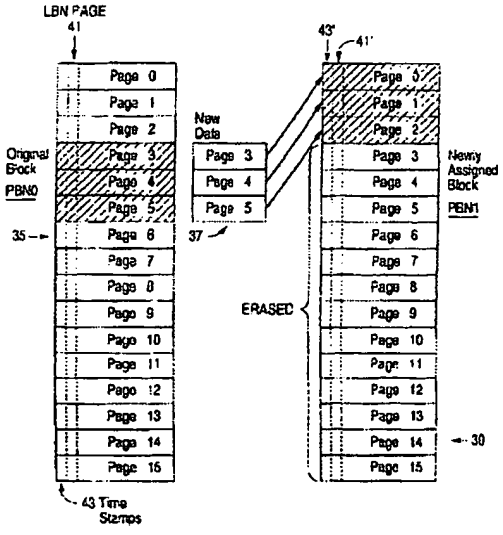
(71)出願人　592012513
サンディスク　コーポレイション
ＳａｎＤｉｓｋ　Ｃｏｒｐｏｒａｔｉｏｎ
アメリカ合衆国、９４０８９、カリフォルニア州、サニーベール、カスピアン　コート　１４０
(74)代理人　100075144
弁理士　井ノ口　壽
(72)発明者　コンレー，ケビン　エム.
アメリカ合衆国、９５１２０、カリフォルニア州、サン　ホセ、アルバラド　コート　５９８３

Ｆターム(参考)　5B025 AD04 AD08 AE05 AE08
5B060 AA06 AA08 AA14

最終頁に続く

(54)【発明の名称】不揮発性メモリにおける部分的ブロックデータのプログラミング動作および読出し動作

(57)【要約】
同じブロックまたは別のブロックのいずれかのブロックの未使用ページに新しいデータをプログラムすることにより、不揮発性メモリブロックのページのすべてよりも少ないページ数のデータが更新される。無変化のデータページを新しいブロックの中へコピーしたり、置換されたデータページの中へフラグをプログラムしたりする必要がないようにするために、この新しいデータページは、置換されたデータページと同じ論理アドレスにより特定され、各ページがいつ書込まれたかを記すタイムスタンプが付加される。上記データを読出すとき、最新のデータページが使用され、旧い置換されたデータページは無視される。この技法は、上記ユニットのうちの１ユニット内の単一の未使用ブロックに更新されるすべてのページを向けることにより、メモリアレイのいくつかの異なるユニットの各々から得られる１ブロックを含むメタブロックにも適用される。



【特許請求の範囲】
【請求項１】
原データと置換データとを不揮発性メモリシステムに同時に格納する方法において、
前記原データと置換データとを同じ論理アドレスにより特定するステップと、
前記原データと置換データとをメモリの中へプログラムした相対的時刻を追跡することにより、前記原データと前記置換データとを識別するステップと、
を有することを特徴とする方法。
【請求項２】
原データと置換データとを不揮発性メモリシステムに格納し、検索する方法において、
前記原データ部と前記置換データ部とを同じ論理アドレスにより特定するステップと、
データ部をメモリの中へプログラムした順序とは逆の順序で該データ部を読出すステップと、
前記データを読出す順序により、同じ論理アドレスを持つ原データ部と置換データ部とを識別するステップと、
を有することを特徴とする方法。
【請求項３】
個々に組織化されて複数ページのメモリ記憶エレメントの中へ入れられる複数ブロックのメモリ記憶エレメントを持つ不揮発性メモリシステムにおいて、前記複数ブロックのうちの１ブロックの少なくとも１ページ内で置換されたデータの代わりに新しいデータを代替し、一方、前記１ブロックの少なくとも別のページ内のデータは置換しないようにする方法において、
前記新しいデータをプログラムして、前記複数ブロックのうちの前記ブロックまたは別のブロックの少なくとも１ページの中へ前記新しいデータを入れるステップと、
前記少なくとも１ページの置換されたデータと前記少なくとも１ページの新しいデータとを共通の論理アドレスにより特定するステップと、
前記新しいデータと前記置換されたデータとの相対的プログラミング時刻を記録するステップと、
を有することを特徴とする方法。
【請求項４】
前記新しいデータと置換されたデータとがプログラムされる個々のページについての相対的プログラミング時刻を記録し、前記新しいページの記録された相対的プログラミング時刻により、前記少なくとも１ページの新しいデータを前記少なくとも１ページの置換されたデータと識別可能にすることを特徴とする請求項３記載の方法。
【請求項５】
前記個々のブロックについての相対的プログラミング時刻を記録し、それによって、共通の論理アドレスを持つデータを含む個々のブロックのプログラミングを行う順序を特定し、さらに、指定した順序で前記個々のブロック内のページをプログラムすることにより、前記ブロック内における前記ページの相対位置によって、前記新しいデータページを個々のブロック内の置換されたデータページと識別可能にすることを特徴とする請求項３記載の方法。
【請求項６】
前記置換されたデータの代りに前記新しいデータを代替するステップの一部として、置換されなかった前記１ブロックの少なくとも別のページ内のデータを前記ブロックまたは別のブロックの中へコピーしないことを特徴とする請求項３記載の方法。
【請求項７】
前記置換されたデータの代りに前記新しいデータを代替するステップの一部として、前記少なくとも１ページの置換されたデータの中へ何も書込まないことを特徴とする請求項３

が、前記新しいデータと置換されたデータとがプログラムされた各時刻におけるクロックの値を格納するステップを含むことを特徴とする請求項４記載の方法。

【請求項９】

前記新しいデータと置換されたデータとの相対的プログラミング時刻を記録するステップが、前記新しいデータと置換されたデータとがプログラムされた各時刻において異なる一連の番号値を格納するステップを含むことを特徴とする請求項４記載の方法。

【請求項１０】

前記新しいデータと置換されたデータとの相対的プログラミング時刻を示す値を格納するステップが、前記値が関連する前記新しいデータおよび置換されたデータと同じページ内に前記個々の値を格納するステップを含むことを特徴とする請求項８または９のいずれか記載の方法。

【請求項１１】

前記複数ブロックのうちの前記１ブロックまたは別のブロックの少なくとも１ページの中へ前記新しいデータをプログラムするステップが、前記ブロックまたは別のブロック内の前記第１の利用可能な未使用ページの中へ前記新しいデータを予め設定した順序でプログラムするステップを含むことを特徴とする請求項３記載の方法。

【請求項１２】

前記少なくとも１ページの置換されたデータと前記少なくとも１ページの新しいデータとを共通の論理アドレスにより特定するステップが、オーバヘッドデータとして前記論理アドレスの少なくとも一部を前記個々のページに記録するステップを含むことを特徴とする請求項３記載の方法，

【請求項１３】

揮発性メモリ内にテーブルを構成するステップを含むステップが、前記共通の論理アドレスに対応する複数の物理ブロックアドレスを含むことを特徴とする請求項１２記載の方法。

【請求項１４】

請求項４に基づいて更新されたデータを読出す方法において、

前記１ブロックからデータページを読出すステップであって、もし新しいデータが前記別のブロックの中へプログラムされていた場合には、前記別のブロックを読出すステップと、

同じ論理アドレスを持つ任意の複数のデータページを特定するステップと、

同じ論理アドレスを持つ任意のページのうちの最新のページを特定するために、前記新しいデータと置換されたデータとの記録された相対的プログラミング時刻を利用するステップと、

更新されていない前記１ブロックの前記少なくとも別のページ内のページと共に、同じ論理アドレスを持つ任意のページの前記最新のページにデータを組み立てるステップと、

を有することを特徴とする方法。

【請求項１５】

請求項５に基づいて更新されたデータを読出す方法において、

前記１ブロック内でデータページを読出すステップであって、もし新しいデータが前記１ブロックの中へプログラムされていた場合には、前記データページがプログラムされた順序と逆の順序で別のブロックを読出すステップと、

すでに読出されたデータを含むページと同じ論理ページアドレスを持つ前記のように読出されたいずれのデータページも無視するステップと、

を有することを特徴とする方法。

【請求項１６】

３以上の格納状態を持つ前記個々のメモリ記憶エレメントを動作させ、それによって各記

【請求項１７】
３以上の格納状態を持つ前記個々のメモリセルの記憶エレメントを動作させ、それによって各記憶エレメントに２ビット以上のデータを格納するステップをさらに有することを特徴とする請求項３～９のいずれか記載の方法。
【請求項１８】
前記記憶エレメントが、個々のフローティングゲートを備えることを特徴とする請求項１７記載の方法。
【請求項１９】
前記不揮発性メモリシステムが、ホストシステムと作動可能に接続する１つの端縁に沿った電気的コネクタを備えた封止されたカード内に形成されることを特徴とする請求項３～９のいずれか記載の方法。
【請求項２０】
少なくとも２つのサブアレイに組織化されるメモリ記憶エレメントのアレイを備えた不揮発性メモリシステムを動作させる方法であって、前記個々のサブアレイが複数の非オーバーラップブロックの記憶エレメントに分画され、１ブロックが一括して消去可能なメモリ記憶エレメントの最小グループを含み、さらに、前記個々のブロックが複数ページの記憶エレメントに分画され、１ページが一括してプログラム可能なメモリ記憶エレメントの最小グループとなるように構成される方法において、
前記少なくとも２つのサブアレイの個々のサブアレイから得られる少なくとも１ブロックをリンクしてメタブロックを形成するステップであって、メタブロックの構成ブロックを１ユニットとして一括消去するステップと、
更新されたデータが格納されているサブアレイにかかわりなく、前記サブアレイのうちの指定したサブアレイ内のみの少なくともう１つのブロック内のページの中へ置換データをプログラムすることにより、前記ブロック内のすべてのページよりも少ない数の前記メタブロックの構成ブロックのうちのいずれかのブロック内で原データのページを更新するステップと、
を有することを特徴とする方法。
【請求項２１】
前記原データと置換データとを格納するステップが、
同じ論理アドレスにより前記原データと置換データとを前記メモリシステムに対して特定するステップと、
前記原データと置換データとが前記メモリのそれぞれのページでプログラムされた相対的時刻を追跡することにより前記原データと前記置換されたデータとを識別するステップと、
を含むことを特徴とする請求項２０記載の方法。
【請求項２２】
不揮発性メモリシステムにおいて、
記憶エレメントのブロック内で組織化された不揮発性メモリ記憶エレメントのアレイであって、消去可能な最小グループの記憶エレメントを個々のブロックが含むように構成されるアレイと、
第２のブロックに格納された原データのすべてよりも少ないデータの更新バージョンであって、前記更新バージョンの後での書込みを示す表示と共に、第１のブロックの中へ該更新バージョンを書込むプログラミングメカニズムと、
前記原データと前記更新バージョンの双方のアドレス指定を同じアドレスを用いて論理的に行うアドレス指定メカニズムと、
前記更新バージョンの後での書込みを示す前記表示によって、前記相対的時刻により少なくとも部分的に前記原データと前記更新バージョンとを識別する読出しメカニズムと、

【発明の属する技術分野】
本発明は、半導体不揮発性データ格納システムのアーキテクチャの技術分野および上記アーキテクチャの動作方法に関し、フラッシュ型の電気的に消去可能でプログラム可能なリードオンリーメモリ（ＥＥＰＲＯＭ）に基づくデータ格納システムに適用される。
【０００２】
【従来の技術】
フラッシュＥＥＰＲＯＭデバイスの一般的利用として、電子デバイス用大容量データ記憶のサブシステムとしての利用がある。このようなサブシステムは、複数のホストシステムの中へ挿入可能な取り外し可能なメモリカードとして、または、ホストシステム内の取り外し不可能な組み込み型記憶装置としてのいずれかの形で一般に具現化される。上記双方の実装構成にはサブシステムに１以上のフラッシュデバイスが含まれ、またサブシステムコントローラが含まれる場合も多い。
【０００３】
フラッシュＥＥＰＲＯＭデバイスは、トランジスタセルからなる１以上のアレイから構成され、各セルは１以上のビットデータの不揮発性格納を行う能力を有する。したがって、フラッシュメモリは、その中でプログラムされたデータの保持用電力を必要としない。しかし、一旦セルをプログラムすると、新しいデータ値での再プログラムが可能となる前にセルの消去が必要となる。これらのセルアレイはグループに分割され、読出し、プログラム、消去の各機能の効率的実装構成が提供される。大容量記憶用の代表的なフラッシュメモリ用アーキテクチャは大きなグループのセルが消去可能なブロックとなるように構成され、１ブロックには一度に消去可能な最小数のセル（消去単位）が含まれる。
【０００４】
１つの市販の形態では、１セクタのユーザデータとユーザデータおよび／またはユーザデータが格納されているブロックと関連するいくつかのオーバヘッドデータを格納できる十分なセルが各ブロックに含まれている。１セクタに含まれるユーザデータの量は、このようなメモリシステムの１つのクラスでは標準の５１２バイトであるが、何らかの別のサイズにすることも可能である。個々のセルブロックを個々に消去可能にするために必要な個々のセルブロックを相互に絶縁するためのスペースが集積回路のチップ上にとられるため、別のクラスのフラッシュメモリではブロックをかなり大きなものにしている。そのためこのような絶縁を行うためのスペースが少なくなる。しかし、さらに小さなセクタ内でのユーザデータの処理も求められるため、大きなブロックの各々が、ユーザデータの読出しとプログラミングの基本単位（プログラミングおよび／または読出し単位）である個々にアドレス可能なページにさらに分割される場合が多い。各ページは通常１セクタのユーザデータを格納するが、１ページが部分セクタまたは複数セクタを格納することもできる。本願では、“セクタ”という用語は、１単位としてホスト間を往来して転送されるユーザデータの量を意味するために用いられる。
【０００５】
大きなブロックシステム内のサブシステム用コントローラは、メモリサブシステムがホストから受信する論理アドレス（ＬＢＡ）と、メモリセルアレイ内の物理ブロック数とページアドレスとの間の変換機能を含むいくつかの機能を実行する。この変換は、論理ブロック数（ＬＢＮ）と論理ページのための中間項の使用を伴うことが多い。また、上記コントローラは、インタフェースバスを介してフラッシュメモリデバイスへ出される一連のコマンドを通じて低レベルのフラッシュ回路の動作の管理も行う。上記コントローラが行う別の機能として、誤り訂正符号（ＥＣＣ）などを利用する種々の手段によって、上記サブシステムに対して格納されたデータの完全性を保持する機能がある。
【０００６】
理想的ケースでは、１ブロックのすべてのページ内のデータは、まだ割り当てられていな

べてよりも少ない複数のページに格納されたデータを更新する必要がある場合の方がさらに一般的である。所定のブロックの残りのページに格納されたデータは変化しないまま残る。１ブロック当たりで格納されるデータのセクタ数が多くなるシステムではこのようなことが生じる確率が高くなる。このような部分的ブロックの更新を行うために現在用いられている１つの技法として、更新対象ページのデータを対応する数の未使用の消去されたブロックのページの中へ書込み、次いで、原ブロックから新しいブロックのページの中へ上記無変化のページをコピーする技法がある。次いで、原ブロックの消去を行い、未使用ブロックのインベントリにこのブロックを追加することができる。この未使用ブロックで後程データのプログラムを行うことができる。別の技法では、更新されたページの同様の書込みを新しいブロックに対して行い、原ブロック内のページのフラグの変更により他のデータページを新しいブロックの中へコピーすることは不要となる。上記原ブロック内のページが更新され、該ページが旧いデータを含むことが示される。次いで、データが読出されると、新しいブロックのページから読出された更新されたデータが、旧いデータとしてフラグされていない原ブロックのページから読出された無変化のデータと組み合わされる。

【０００７】

（発明の開示）

端的にかつ一般的に述べると、本発明の１つの主要な局面によれば、１ブロック内のページのすべてよりも少ないデータの更新を行う場合、原ブロックから新しいブロックへの無変化のデータのコピーと原ブロック内でのフラグの更新の必要性の双方が回避される。これは、置換されたデータページと更新されたデータページの双方のページを共通の論理アドレスを用いて保持することにより達成される。次いで原ページと更新されたデータページとは、これらのページがプログラムされた相対的順序により識別される。読出しが行われている間、同じ論理アドレスを持つページに格納された最新のデータが無変化のデータページと組み合わされ、一方、更新されたページの原バージョン内のデータは無視される。原データとは異なるブロック内のページ、または、同じブロック内の利用可能な未使用ページのいずれかのページへ上記更新データの書込みを行うことができる。１つの具体的な実装構成では、同じ論理アドレスを持つページが書込まれた相対的順序の決定を可能にする各データページと共に、あるタイムスタンプ形式が格納される。別の具体的な実装構成では、ブロック内である特定の順序でページのプログラミングを行うシステムにおいて、あるタイムスタンプ形式が各データブロックと共に格納され、ブロック内の最新ページのコピーがブロック内の該ページの物理位置により確定される。

【０００８】

これらの技法により、原ブロックから新しいブロックへ無変化のデータをコピーする必要性と、更新されたデータを含む原ブロックのページ内でフラグまたは別のデータの変更を行う必要性とが回避される。置換ページ内でフラグまたは他のデータの変更を行う必要がなくなることにより、当該同一ブロックの隣接ページ内で前回書込まれたデータが妨害を受ける可能性が除外される。上記妨害はこのような書込み動作から生じる可能性がある。また、追加のプログラム処理のパフォーマンスペナルティも回避される。

【０００９】

上記要約された技法と関連して利用することが可能な別の動作特性により、個々のメモリセルブロック内の個々のデータページの論理オフセット値が絶えず追跡され、それによって置換されたデータと同じ物理ページオフセット値と共に更新されるデータを格納する必要がなくなる。これによって新しいブロックのページのさらに効率的な利用が可能となり、置換されたデータと同じブロックの任意の消去されたページに更新データを格納することさえ可能となる。

【００１０】

ープを“メタブロック”と呼ぶ。メタブロックの構成ブロックは、単一のメモリ集積回路チップ上にすべて配置されるか、２以上の異なるチップ上に配置されて、２以上のこのようなチップを使用するシステム内に存在するかのいずれかとなる。これらのブロックのうちの１ブロックのページのすべてよりも少数のページ内のデータが更新されると、当該同一ユニット内の別のブロックの使用が通常要求される。実際、上述の技法またはその他の技法をメタブロックの各ブロックで別個に採用することも可能である。したがって、メタブロックの２以上のブロックのページ内のデータを更新するとき、２以上の追加ブロック内のページを使用する必要がある。例えば、上記メタブロックを形成する４つの異なるメモリユニットからなる４つのブロックがある場合、例えば、追加の４ブロックにまで、これらユニットの各ユニット内の１つのブロックが原ブロックの更新ページを格納するために使用される或る確率が生じる。原メタブロックの各ブロックについて各ユニットで１つの更新ブロックが潜在的に要求される。さらに、本発明によれば、上記メタブロック内の２以上のブロックのページから更新されたデータを上記ユニットのうちのただ１つのユニット内の共通ブロックのページに格納することができる。これにより、更新データの格納に必要な未使用の消去されたブロックの数が大幅に減少し、それによってデータの格納に利用可能なメモリセルブロックの利用がさらに効率的になる。メモリシステムがメタブロックからの単一ページを更新する頻度が高い場合、この技法は特に有用である。

【００１１】

本発明の追加の局面、特徴および利点は、実施形態例についての以下の説明の中に含まれ、添付図面と関連して読まれることが望ましい。

【００１２】

【発明の実施の形態】

現行の大ブロック管理技法についての説明

図１は、一般的なフラッシュメモリデバイス内部のアーキテクチャを示す。その主要な特徴として、外部コントローラとのインタフェースを行うための入出力（Ｉ／Ｏ）バス４１１と制御信号４１２と、コマンド信号、アドレス信号および状態信号用のレジスタを用いて内部メモリの動作を制御するメモリ制御回路４５０が含まれる。フラッシュＥＥＰＲＯＭセルからなる１以上のアレイ４００が含まれ、各アレイは、アレイ自身の行デコーダ（ＸＤＥＣ）４０１および列デコーダ（ＹＤＥＣ）４０２、１グループのセンス増幅器およびプログラム制御回路構成（ＳＡ／ＰＲＯＧ）４５４とデータレジスタ４０４を備えている。現在、メモリセルには記憶エレメントとして１以上の導電性フローティングゲートが通常含まれているが、この代わりに別の長時間電子電荷記憶エレメントを使用してもよい。各記憶エレメントについて設定される２つのレベルの電荷を用いてメモリセルアレイを動作させることも可能であり、したがって各エレメント当り１ビットのデータが格納される。上記とは別に、各記憶エレメントと関連して３以上の記憶状態を設定してもよい。その場合、１ビット以上のデータが各エレメントに格納される。

【００１３】

所望の場合、例えば、１９９９年３月３０日発行され、本願の譲受人であるサンディスクコーポレイションに譲渡された米国特許第５，８９０，１９２号に教示されているように、関連するＸデコーダ、Ｙデコーダ、プログラム／検証された回路、データレジスタ等と一体に複数のアレイ４００が設けられる。上記特許は本願明細書で上記参照により援用されている。メモリシステムの特徴に関連して、Ｋｅｖｉｎ Ｃｏｎｌｅｙらによる２０００年２月１７日出願の同時継続中の特許出願第０９／５０５，５５５号に記載がある。上記特許は、本願明細書で上記参照により明白に援用されている。

【００１４】

外部インタフェースＩ／Ｏバス４１１と制御信号４１２とは以下を含むことができる。

CS－Chip選択	フラッシュメモリインタフェースの起動に使用される。
RS－READストローブ	メモリアレイからのデータ転送にI／Oバスが使用中であることを示すために使用される。
WS－Writeストローブ	メモリアレイへのデータ転送にI／Oバスが使用中であることを示すために使用される。
AS－Addressストローブ	アドレス情報の転送にI／Oバスが使用中であることを示す。
AD［7：0］－アドレス／データバス	このI／Oバスは、コントローラとメモリ制御４５０のフラッシュメモリコマンド用レジスタ、アドレスレジスタおよびデータレジスタ間でのデータ転送に使用される。

【００１５】
このインタフェースは一例として示すものにすぎず、同じ機能を提供する別の信号構成の利用も可能である。図１には、フラッシュメモリの関連する構成要素を備えた一つのフラッシュメモリアレイ４００しか示されていないが、単一のフラッシュメモリチップ上に多数の同様のアレイの存在が可能である。これらのアレイは、共通のインタフェースとメモリ制御回路構成とを共有しながら、個々に独立したＸＤＥＣ、ＹＤＥＣ、ＳＡ／ＰＲＯＧおよびＤＡＴＡ　ＲＥＧ回路構成を備えて、読出し動作とプログラム動作の同時動作を可能にすることを目的とするものである。
【００１６】
Ｉ／ＯバスＡＤ［７：０］４１１と結合しているデータレジスタを介してデータはメモリアレイからデータレジスタ４０４を通って外部コントローラへ転送される。データレジスタ４０４は、センス増幅器／プログラミング回路４５４とも結合される。各センス増幅器／プログラミング回路エレメントと結合されたデータレジスタのエレメントの数は、各メモリセルの各記憶エレメントに格納されているビット数により決めてもよい。フラッシュＥＥＰＲＯＭセルの各セルには記憶エレメントとして１以上のフローティングゲートが含まれている。多状態モードでメモリセルの処理を行う場合、各記憶エレメントは２または４などの複数ビットを格納するものであってもよい。上記とは別に、メモリセルは、２進モードで処理して１記憶エレメント当たり１ビットデータの格納を目的とするものであってもよい。
【００１７】
行デコーダ４０１は、アクセス対象の物理ページを選択するためにアレイ４００に関連する行アドレスの復号化を行う。行デコーダ４０１は、メモリ制御論理回路４５０から内部行アドレスライン４１９を介して行アドレスを受け取る。列デコーダ４０２はメモリ制御論理回路４５０から内部列アドレスライン４２９を介して列アドレスを受け取る。
【００１８】
図２は、代表的な不揮発性データ格納システムのアーキテクチャを示す。このケースでは、格納メディアとしてフラッシュメモリセルが採用されている。１つの形態では、このシステムは、一方の側部に沿って延在する電気的コネクタを備えた取り外し可能なカード内にカプセル化され、このカードがホストの差込み口の中へ挿入されると、ホストとのインタフェースが行われる。上記とは別に、図２のシステムは、永久にインストールされた組み込み型回路などの形でホストシステムの中へ組み込まれる場合もある。このシステムでは、単一のコントローラ３０１が利用され、このコントローラ３０１によりハイレベルのホスト機能とメモリ制御機能とが実行される。フラッシュメモリメディアは１以上のフラ

ラッシュメモリアレイ間を往来する転送データのロードを行うことが可能となる。コントローラ３０１によりホストシステム（図示せず）とのインタフェースが行われ、このホストシステムを用いてフラッシュメモリアレイ間を往来するユーザデータの転送が行われる。図２のシステムがカード内に含まれているケースでは、ホストインタフェースには、カードとホスト装置とつながる接続プラグとソケットアセンブリ（図示せず）とが含まれる。

【００１９】

コントローラ３０１は、ホストからコマンドを受け取り、ある特定の論理アドレスから始まるユーザデータの１以上のセクタの読出しあるいは書込みを行う。このアドレスは、物理メモリセルブロックの境界と整合する場合もあれば、整合しない場合もある。

【００２０】

上述したように、複数ページに分割された大容量メモリセルブロックを含む従来技術によるシステムのなかには、原ブロックからホストにより書込まれた新しい更新データを含む新しいブロックへ更新されないブロックから得られるデータのコピーを行わなければならないものもある。この技法が図４に示されており、多数のメモリブロックのうち２つのブロックが含まれている。一方のブロック１１（ＰＢＮ０）は８ページに分割されて示され、このブロックのページの各ページに１セクタのユーザデータが格納されるようになっている。各ページ内に含まれるオーバヘッド用データフィールドにはブロック１１のＬＢＮを含むフィールド１３が含まれる。１論理ブロック内での論理ページの順序は、物理ブロック内の対応する物理ページに関して固定されている。第２の同様に構成されたブロック１５（ＰＢＮ１）は未使用の消去されたブロックのインベントリから選択される。原ブロック１１のページ３～５内のデータは、新しい３データページ１７により更新される。新しいデータが新しいブロック１５の対応するページ３～５の中へ書込まれ、ブロック１１のページ０～２、６、７からのユーザデータは新しいブロック１５の対応するページの中へコピーされる。新しいブロック１５のすべてのページがプログラミング動作の単一シーケンスの形で好適にプログラムされる。ブロック１５のプログラム後、原ブロック１１を消去して、後で利用するためにインベントリの中に入れておくことができる。ブロック１１と１５との間のデータのコピーは、原ブロック内の１以上のページからのデータの読出しと、新しく割り当てられたブロックのページに対する同じデータの後でのプログラミングを必要とするものであるが、このコピーにより、書込みパフォーマンスが大幅に低下し、格納システムの使用可能な耐用期間が大幅に短くなる。

【００２１】

図５Ａと５Ｂを参照すると、図４との関連で説明したデータの更新前（図５Ａ）とデータの更新後（図５Ｂ）に、部分的テーブルによる論理ブロックの対応付けが原ブロック１１と新しい物理ブロック１５とに変換されて示されている。この例では、データ更新前に、ＬＢＮ０のページ０～７が原ブロック１１によりＰＢＮ０の対応するページ０～７の中へ格納される。このデータ更新後、ＬＢＮ０のページ０～７が新しいブロック１５によりＰＢＮ１の対応するページ０～７に格納される。次いで、ＬＢＮ０からデータの読出しを求める要求の受信が物理ブロック１１の代わりに物理ブロック１５へ向けられる。一般的コントローラの動作では、データフィールド１３の読出し時にアドレス指定されるＰＢＮの物理ページと情報とから読出されるＬＢＮフィールド１３から、図５Ａと図５Ｂに示すテーブルの形のテーブルが構成される。このテーブルは、アクセスを容易にするためにコントローラの揮発性メモリの中に通常格納されるが、一般に、システム全体に関連する完全なテーブルの一部が任意の１時点に格納されるにすぎない。このテーブルの一部は、このテーブル部分に含まれるブロックに関わる読出し動作またはプログラミング動作の直前に通常形成される。

【００２２】

従来技術による別のシステムでは、ユーザデータと共にフラグがページに記録され、これ

クへ書込まれる。したがって、書込み動作に関与しないが、置換されたデータと同じ物理ブロック内に含まれるこのブロックのページ内のデータを新しいブロックの中へコピーする必要はない。この処理が図６に示され、原ブロック２１（ＰＢＮ０）内のデータページ３～５が再び更新されている。データ２３からなる更新ページ３～５が新しいブロック２５の対応するページ内へ書込まれる。同じ処理の一部として、新／旧フラグ２７がページ３～５の各々の中へ書込まれ、当該ページのデータが旧いデータであることが示されているが、一方、残りのページ０～２、６、７用のフラグ２７は“新”にセットされた状態のままである。同様に、新しいＰＢＮ１がブロック２１のページ３～５の各々別のオーバヘッドデータフィールドの中へ書込まれ、更新データがどこに位置するかが示される。ＬＢＮとページとは物理ページの各範囲内のフィールド３１に格納される。

【００２３】

図７Ａと７Ｂは、データの更新完了前（図７Ａ）とデータの更新完了後（図７Ｂ）のデータＬＢＮ／ページとＰＢＮ／ページとの間の対応関係を示すテーブルである。ＬＢＮの無変化のページ０～２，６と７はＰＢＮ０の中へ対応づけられた状態のまま残り、一方、更新されたページ３～５はＰＢＮ１内に存在することが示される。図７Ｂのテーブルは、ブロックＰＢＮ０内のページのオーバヘッドデータフィールド２７、２９、３１をデータ更新後に読出すことによりメモリコントローラにより構成される。フラグ２７が、原ブロックＰＢＮ０のページ３～５の各ページで“旧”にセットされているため、当該ブロックは、当該ページに関連するテーブルに二度と現れることはない。逆に、新しいブロック番号ＰＢＮ１が、更新ページのオーバヘッドフィールド２９’から読出された後、代わりに出現する。データがＬＢＮ０から読出されると、図７Ｂの右欄にリストされたページに格納されたユーザデータが読出され、次いで、図示された順序で組み立てられてホストへ転送される。

【００２４】

一般に種々のフラグが、ＬＢＮやＥＣＣなどの関連する別のオーバヘッドデータと同じ物理ページに配置される。したがって、データがすでに置換されているページに新／旧のフラグ２７およびその他のフラグをプログラムすることは、１ページが複数のプログラミングサイクルをサポートすることを必要とする。すなわち、メモリアレイは、消去が行われる間にそのページが少なくとも２段階のプログラミングを行い得る能力を備えていなければならない。さらに、より高いオフセット値またはアドレスを持つブロック内の別のページが予めプログラムされているとき、ページをプログラムする能力がブロックによりサポートされている必要がある。しかし、物理的にシーケンシャルな方法でしかブロック内のページをプログラムできない旨の指定により設けられた制限に起因して、フラッシュメモリによってはこのようなフラグの利用ができないものもある。さらに、ページによっては有限回のプログラムサイクルしかサポートされていないものもあり、場合によっては、プログラムされたページの追加プログラミングが許されていない場合もある。

【００２５】

求められるメカニズムは、無変化のデータの現行ブロックからのコピー、あるいは、予めプログラムされたページに対するフラグのプログラミングのいずれも行わずに、現行のブロックに格納されたデータを部分的に置き換えるデータ書込みを行うことが可能なメカニズムである。

【００２６】

本発明の実施形態例についての説明

多くの異なるタイプのフラッシュＥＥＰＲＯＭが存在するが、これらの各ＥＥＰＲＯＭには、狭い集積回路の面積に形成される高いパフォーマンスを示すメモリシステムを動作させるために避ける必要があるそれ自身の制限が存在する。上記タイプのＥＥＰＲＯＭのなかには、予めプログラムされたページの中へデータの書込みを全く行わないものもあるた

ている同じブロックの他のページ内のデータが妨害を受ける可能性がある。

【００２７】

上記の事実が問題となることが判明したメモリシステムの一例として、ビットラインと共通電位との間で連続する回路列としてメモリセルの列が形成されるＮＡＮＤ型メモリシステムがある。各ワードラインは、上記のような各回路列内の１つのセルから形成されるメモリセルの行の両端にわたって延在する。多状態モードでこのようなメモリを処理して、上記のような各セルに２ビット以上のデータを格納する場合、上記メモリは、特にメモリ状態に対する上記のような妨害を受けやすくなる。メモリセルトランジスタのしきい値電圧の範囲の利用可能なウィンドウは上記のような処理により非オーバラップ電圧レベルの狭い範囲に分割されるが、上記各範囲はレベル数、したがって、各セルに格納されているビット数の増加につれてより狭くなる。例えば、４つのしきい値範囲を用いる場合、２ビットのデータが各セルの記憶エレメントに格納される。さらに、４つのしきい値電圧の各範囲が必然的に狭いため、同じブロック内の別のセルのプログラミングによりセル状態が妨害を受ける可能性は多状態処理に伴って大きくなる。この場合、図６および図７Ａと７Ｂとに関連して説明したような新／旧または別のフラグの書込みを許容することはできなくなる。

【００２８】

図４～７Ｂとに関連して上述した現行の各メモリ管理技法の一般的特徴として、論理ブロック数（ＬＢＮ）とページオフセット値とが、システム内で多くて２つの物理ブロック数（ＰＢＮ）に対応づけられるという点が挙げられる。一方のブロックは原ブロックであり、他方のブロックは更新されたページデータを含むブロックである。データは、上記ブロックの論理アドレス（ＬＢＡ）の下位ビットに対応するブロック内のページ位置へ書込まれる。この対応付けにより、種々のタイプのメモリシステムでの固有の特徴が示される。以下で説明する技法では、置換されたデータを含むページと同じＬＢＮとページオフセット値とが更新データを含むページに割り当てられる。しかし、置換した形で原データを含むページのタグ付けを行う代わりに、メモリコントローラは、この置換されたデータを含むページを新しい更新バージョンを含むページと識別する。この識別は、（１）カウンタなどを用いて追跡することにより、同じ論理アドレスを含むページが書込まれる順序の識別を行う、および／または、（２）ブロック内で最下位のページアドレスから最高位のページアドレスへページが順に書込まれるとき、より高位の物理アドレスに最新のデータコピーが含まれる物理ページアドレスと更新バージョンを含むページとの識別を行うかのいずれかにより行われる。したがって、読出しのためにデータのアクセスを行うとき、同じ論理アドレスを持つ置換されたデータが含まれるページが存在する場合には最新のページ内のデータが使用され、一方、置換されたデータは無視される。

【００２９】

図８と図９に関連して上記技法の第１の具体的な実装構成について説明する。この例でも状況は図４～７Ｂに関連して説明した従来の技術の状況と同じである。すなわち、ブロック３５内でデータの部分的再書込みが行われる。但し、この例では各ブロックが１６ページを含むように示されている。ブロック３５（ＰＢＮ３５）のページ３～５の各々に関連する新しいデータ３７が、上述の場合と同様に、予め消去された新しいブロック３９（ＰＢＮ１）の３ページの中へ書込まれる。ＬＢＮおよび更新データを含むＰＢＮ１のページの中へ書込まれるページオフセット値のオーバヘッドデータフィールド４１は、最初のブロックＰＢＮ０内の置換されたデータのページのオーバヘッドデータフィールドと同じである。フィールド４１と４１’内のデータから形成される図９のテーブルに上記のことが示されている。第１列内の論理ＬＢＮとページオフセット値とが対応づけられて、第２列内の第１の物理ブロック（ＰＢＮ０）の中へ入れられ、次いで、上記論理ＬＢＮとページオフセット値は、更新されたページについては、第３列内の第２の物理ブロック（ＰＢＮ

る。

【００３０】

同じＬＢＮとページオフセット値とを持つ２つのページのうちいずれのページが更新データを含むかを決定するために、各ページは別のオーバヘッドフィールド４３を含み、このオーバヘッドフィールドによって、同じ論理アドレスを持つ別のページのプログラミング時刻に対する上記各ページの少なくとも相対的なプログラミング時刻の表示が行われる。これによって、メモリからデータを読出す際に、コントローラは、同じ論理アドレスが割り当てられているデータページの相対的作成時期の決定を行うことが可能となる。

【００３１】

タイムスタンプ形式を含むフィールド４３に書込みを行ういくつかの方法がある。最も単純な方法としては、当該フィールドの関連ページのデータをプログラムする際に、システム内のリアルタイムのクロック出力を当該フィールドに記録するという方法がある。その場合、同じ論理アドレスを持つ後でプログラムされたページには、フィールド４３に記録された後の時刻が出力される。しかし、このようなリアルタイムのクロックがシステムで利用できない場合、別の技法を利用することができる。１つの具体的な技法として、フィールド４３の値としてモジュローＮカウンタの出力を格納する方法がある。このカウンタ値の範囲は、同じ論理ページ番号を用いて格納することが意図されているページ数よりも１だけ大きい値にすることが望ましい。例えば、原ブロックＰＢＮ０内のある特定ページのデータを更新するとき、コントローラは、更新データを持つページのフィールド４３に格納されたカウント値をまず読出し、１などのある量だけこのカウント値を増分し、次いで、フィールド４３’として増分された当該カウント値を新しいブロックＰＢＮ１に書込む。このカウンタ値は、Ｎ＋１のカウントに達すると、０へロールオーバーする。同じＬＢＮを持つブロックの数はＮよりも少ないため、格納されたカウント値には常に不連続なポイントが存在する。この不連続ポイントに対する正規化を行って上記ロールオーバーの処理を行うことは容易である。

【００３２】

データの読出し要求が行われると、同じＬＢＡとページオフセット値とを持つページフィールド４３と４３’内のカウント値を比較して、新しいページと置換ページのデータ間の識別がコントローラにより容易に行われる。データファイルの最新バージョンを読出す必要に応じて、特定した新しいページから得られるデータは更新されていない原ページと共に組み立てられ、データファイルの最新バージョンの中へ入れられる。

【００３３】

図８の例では、新しいデータページ３７は、原ブロックＰＢＮ０内で上記新しいデータページにより置き換えられる同じページ３～５に格納されるのではなく、新しいブロックＰＢＮ１の最初の３ページ０～２に格納されることに留意されたい。個々の論理ページ番号を追跡することにより、置換されたデータを含む旧いブロックのページオフセット値と同じ新しいブロックのページオフセット値に上記更新データを必ずしも格納する必要はない。置換されたデータのページと同じブロックの消去されたページへ更新されたデータページを書込むことも可能である。

【００３４】

この結果、どの物理ページの中へ新しいデータの書込みが可能かについて制限を設けるような上述した技法が示す制約は存在しなくなる。しかし、これらの技法が実行されるメモリシステムに若干の制約が設けられる場合がある。例えば、１つのＮＡＮＤシステムでは、シーケンシャルな順序でブロック内のページをプログラムすることが求められる。これは、新しいブロック２５で行われたような（図６）中程のページ３～５のプログラミングによって、後でプログラムを行うことができないページ０～２が無駄になることを意味する。新しいブロック３９の利用可能な最初のページの新しいデータ３７をこのような限定的なフラッシュに格納することにより、別のデータを格納するために後で使用する残

～７に新しいデータを格納することが可能となる。これによって、このようなシステムの利用可能な記憶容量が最大限に利用される。

【００３５】

図８のブロックの個々のページに格納されたデータの構造の一例を図１０に示す。最も大きな部分はユーザデータ４５である。ユーザデータから計算された誤り訂正符号（ＥＣＣ）４７もこのページに格納される。ＬＢＮとページタグ４１（論理ページオフセット値）とを含むオーバヘッドデータ４９、タイムスタンプ４３およびオーバヘッドデータから計算されたＥＣＣ５１もこのページに格納される。ユーザデータＥＣＣ４７から独立したオーバヘッドデータをカバーするＥＣＣ５０をオーバヘッド４９が含むことにより、オーバヘッド４９は上記ユーザデータから独立に個々に読出され、ページに格納されたデータのすべての転送を必要とせずに、このオーバヘッド４９を有効なデータとして評価することができる。しかし、上記とは別に、オーバヘッドデータ４９の別個の読出しが頻繁ではないイベントの場合、１ページ内のＥＣＣの総ビット数を減らすために、単一のＥＣＣによりこのページ内のデータのすべてをカバーする場合もある。

【００３６】

発明性を有する上記技法の第２の具体的な実装構成について図８と関連して説明することもできる。本例では、タイムスタンプを用いてブロックに格納したデータの相対的作成時期のみが決定され、一方、同じＬＢＮとページ数とを担持するページのうちの最新ページについては、そのページの相対的物理位置により当該最新ページが決定される。この場合、各ページの一部としてタイムスタンプ４３を格納する必要はない。代わりに、各ブロックについて、該ブロックの一部として単一のタイムスタンプを記録するか、不揮発性メモリ内のどこかに単一のタイムスタンプを記録するかのいずれかが行われ、次いで、ブロックの中へ１データページが書込まれる度にこのタイムスタンプの更新を行うようにしてもよい。次いで、データは、同じＬＢＮを持つデータページを含む最新の更新ブロックの最後のページから始まる降順の物理アドレスの順序でページから読出される。

【００３７】

例えば、図８では、最後のページ（ページ１５）から第１ページ（ページ０）へ新しいブロックＰＢＮ１で上記ページがまず読出され、次いで原ブロックＰＢＮ０のページの同じ逆順で読出しが後続する。一旦新しいブロックＰＢＮ１から論理ページ３、４、５が読出されてしまうと、同じ論理ページ番号により特定される原ブロックＰＢＮ０の当該ページ内の置換されたデータについては読出し処理が行われている間スキップすることができる。詳細には、一旦物理ページ３、４、５のＬＢＮ／ページ４１が、新しいブロックＰＢＮ１からすでに読出されたページのＬＢＮ／ページと同じであることがコントローラにより決定されると、この例では、読出しが行われている間、旧いブロックＰＢＮ０の物理ページ３、４、５はスキップされる。この処理により、読出し速度が上昇し、各ページについて格納の必要があるオーバヘッドビット４９の数を減らすことができる。さらに、この逆順のページ読出し技法を用いると、読出し動作中にコントローラが使用する図９のテーブルを図５Ａと５Ｂの形に単純化することが可能となる。この効率的な読出し処理を実行するためには、共通論理ブロックのデータと物理ブロックがプログラムされた相対的時刻とを含む当該物理ブロックの識別子を知るだけで十分である。

【００３８】

図１１は、ブロックＰＢＮ０に最初に書込まれたデータに対する第２の更新データが含まれる図８の例の拡張を示す図である。論理ページ５、６、７、８に関連する新しいデータ５１が、該データのＬＢＮとページ番号と共に新しいブロックＰＢＮ１のそれぞれの物理ページ３、４、５、６へ書込まれる。この例では、論理ページ５のデータは２度目に更新されたものである点に留意されたい。新しいブロックＰＢＮ１の最後のページから始まる読出し動作の最中に、関心対象データからなる最も新しく書込まれた論理ページ８、７、６、５が上記順序で読出される。その後、ＰＢＮ１の物理ページ２内のＬＢＮ／ペー

理ページ１と０が読出される。次に、物理ページ１５から始まる原ブロックＰＢＮ０のページが読出される。物理ページ１５～９の読出し後、ページ８～３の各々のＬＢＮ／ページフィールドがすでに読出されているデータを持つページのＬＢＮ／ページフィールドに一致し、したがって、当該ページから旧いデータを読出す必要がないことにコントローラは気づく。このようにして読出し処理の効率が改善される。最後に、当該データが更新されなかったため、物理ページ２～０の原データが読出される。

【００３９】

データがページ０から順に消去されたブロックの物理ページ位置に書込まれるため、逆順にページを読出すこの例によって、置換されたデータページから新しいデータページが効率的にソートされることに留意されたい。しかし、この技法は、このようなある特定のプログラミング上の制約を持つメモリシステムの場合の使用に限定されるものではない。ページが所定のブロック内でプログラムされる順序がわかっている限り、当該ページから得られるデータが書込まれた逆順で該データを読出すことも可能である。所望のこととして、以前にプログラムされた他のページと共通のＬＢＮを持つ最新のプログラムされたページがまず読出され、次いで、これらのページが最新のプログラムされたページとなるということが挙げられる。置換されたバージョンを後で容易に特定できるように、更新ページの最新バージョンがまず読出される。

【００４０】

論理データと物理ページアドレス間の対応関係を示す図１１の例に関連するテーブルを図１２に示す。２回のデータ更新が行われたにもかかわらず、双方の更新データとも第２のブロックＰＢＮ１の単一の列により表されている。論理ページ５に関連してＰＢＮ１に書かれている物理ページは、２回目の更新時に、作成される当該ページへ単に変更されるにすぎない。この更新で第３のブロックを用いる場合、この別のブロック用の別の列が追加される。図１２のテーブルは、共通のＬＢＮのデータが書込まれたブロック内の各ページからオーバヘッドデータを読出すことにより構成され、逆方向からのページ読出し技法を利用しないときには、１回目の実装構成でこのテーブルを利用することが可能である。上述した逆方向からのページ読出し技法を利用する場合、図１２のテーブルを構成して、ＬＢＮと当該ＬＢＮのデータを含むすべてのＰＢＮとの間の対応関係を特定する必要がある。

【００４１】

１以上のページの更新が行われた、物理ブロックから読出されたデータページの効率的組織化方法が図１３により例示されている。少なくともいくつかのデータページの、好ましくは最大のデータブロックを一度にバッファできるだけの十分なスペースがコントローラの揮発性メモリの中に設けられる。これが図１３に示されている方法である。不揮発性メモリブロックに格納されている量に等しい１６のデータページがコントローラメモリに格納される。これらのページは一般に順不同に読出されるため、各データページは他のページに関してその正しい位置に格納される。例えば、図１１の逆方向からのページ読出し動作時に、論理ページ８は、この論理ページが最初の読出しページであれば、円で囲んで記した“１”で示すように、コントローラメモリの位置８に格納される。次に論理ページ７等々と処理が行われ、ホストが望むすべてのデータページが読出されて、コントローラメモリに格納されるまで処理が続けられる。次いで、バッファメモリ内のデータの順序を操作する必要なくページデータのセット全体がホストへ転送される。これらのデータページは、コントローラメモリ内のこれらのページの正しい位置への書込みによりすでに組織化されている。

【００４２】

図８と図９とに関連して説明した技法を利用する不揮発性メモリシステムのプログラミング方法が図１４のフローチャートに示されている。ブロック５２により示されるように、更新対象の現行ファイルのページに関連するデータがホストシステムから受信される。格

して１６ページが示されている。格納すべき更新データのページ数がシステムの１ブロックの記憶容量以上であれば、１以上の未使用の消去されたブロックのアドレス指定がステップ５５で行われ、ステップ５７で、新しいデータページがアドレス指定されたブロックへ書込まれる。一般に、１以上のデータブロックの更新により、新しいデータにより置換されたデータを格納する１以上のブロックが結果として生じることになる。その場合、ステップ５９に示されるように、置換されたデータを持つ当該ブロックが消去用として特定される。パフォーマンスを上げるために、消去動作は、バックグラウンドで行われるか、ホストから要求されるプログラミング動作または読出し動作が行われていない時に行われることが望ましい。消去後、ブロックは別途使用するために未使用の消去されたブロックのインベントリへ戻される。上記とは別に、プログラミング動作の必要が生じるまでブロックの消去を延期することもできる。

【００４３】

一方、ステップ５３で、ブロックの最大記憶容量を利用するページよりも新しいデータページ数が少ないと判定された場合、次のステップ６１により、他のデータを用いてプログラムしたいくつかのページを持つ十分な未使用ページが１ブロック内に存在するかどうかの判定が行われる。十分な未使用ページが１ブロック内に存在すれば、ステップ６３でそのようなブロックのアドレス指定が行われる。存在しなければ、ステップ６５で、全く未使用の消去されたブロックのアドレス指定が行われる。いずれの場合も、ステップ６７で、アドレス指定したブロックの未使用ページの中へ新しいデータがプログラムされる。このプログラミング処理の一部として、上述した方法で、ＬＢＮとページオフセット値とがフィールド４１の中へ書込まれ、タイムスタンプが、更新されたデータのページ（図８）の各フィールド４３の中へ書込まれる。

【００４４】

上記プログラミング処理の望ましい特徴として、置換されたデータだけが格納された任意のブロックが将来のプログラミングに利用できるようになることが挙げられる。したがって、データ更新処理の結果、ブロック全体が単に置換されたデータと共に残っているか否かという問いがステップ６９で行われる。残っていれば、このようなブロックはステップ７１で消去用として待ち行列に入れられ、プロセスは完了する。残っていなければ、ステップ７１はスキップされ、データ更新は終了する。

【００４５】

メタブロック処理

プログラミング時間の短縮によりパフォーマンスの改善を図るために、目標として、他のペナルティを受けることなく同時にプログラム可能なできるだけ多くの数のセルをプログラムすることが挙げられる。１つの実装構成では、メモリアレイが、図１５の複数ユニット８０～８３のような実質的に独立しているサブアレイやユニットに分画され、各ユニットは、図に示すように、多数のブロックに分画される。次いで、データページは２以上のユニットに同時にプログラムされる。別の構成では、これらユニットのうちの１以上のユニットが複数のメモリチップからさらに組み合わされる。これら複数のチップは、より高いデータスループットを達成するための（図２に示すような）単一のバスまたは複数の独立バスと接続することも可能である。この拡張として、プログラミング、読出し、消去を一括して行うために、異なるユニットからのブロックをリンクする方法がある。一例を図１５に示す。例えば、ユニット８０～８３のそれぞれのユニットから得られるブロック８５～８８をメタブロックとして一体に動作させることが可能である。上述したメモリの実施形態の場合と同様、メモリアレイの消去可能な最小のグループである各ブロックが複数ページに分画され、１ページにはブロック内で一体にプログラム可能な最小数のセルが含まれる。したがって、図１５に示すメタブロックのプログラミング動作には、通常、メタブロックを形成するブロック８５～８８の各ブロックの少なくとも１ページの中へデータ

クから形成される。

【００４６】

このようなメモリを動作させる過程で、他のメモリの場合と同様、ブロック全体よりも少ないデータページ数を更新する必要が生じる場合が多い。この更新は、図４または図６のいずれかと関連して上述した方法と同様の方法でメタブロックの個々のブロックについて行うこともできるが、好ましくは図８と関連して説明した改善された技法を用いて行うことが望ましい。これら３つの技法のうちのいずれかを用いてメタブロックの１ブロックのデータを更新する場合、同じユニット内の追加のメモリブロックも使用される。さらに、データ更新は、メタブロックのブロックの２以上のページうちの１以上のページに関連する新しいデータの書込みを必要とする場合もある。この書込みには、たとえ数ページ内だけのデータ更新にすぎない場合でも、メタブロックに格納されたデータファイルを更新するために、４までの追加ブロック９０～９３（４つのユニットの各ユニット内で１ブロック）の使用が必要となる場合もある。

【００４７】

このような部分的ブロックの更新に必要なブロック数を減らすために、本発明の別の局面に基づいて、図１６に示されているように、ブロック８０内に未使用ページが残っている限り、図示されたメタブロックのブロックのうちのいずれかのブロック内でメモリユニット８０内の単一の追加ブロック９０に対するデータページの更新が行われる。例えば、３ページのブロック８６と２ページのブロック８８内のデータを一度に更新する場合、５ページ分のすべての新しいデータがブロック９０の中へ書込まれる。これにより１メモリブロックの使用が節減されるため、利用可能な消去されたブロックの数を実際に１ブロック分増やすことが可能となる。このブロックの増加は、消去されたブロックのインベントリが使い果たされるまでの時間の回避または少なくとも延長に役立つ。４つのブロック８５～８８の各々から得られる１以上のページを更新する場合、新しいデータページのすべてが単一ブロック９０の中でプログラムされ、それによって追加の３メモリブロックの更新が回避される。新しいデータページ数が未使用ブロックの容量を上回る場合、ブロック９０が受け入れることができないページは、別の未使用ブロックへ書込まれる。この未使用ブロックは、同じユニット８０または別のユニット８１～８３のうちの１つの中にあるものであってもよい。

【００４８】

様々な実施形態例と関連して本発明を説明してきたが、本発明は添付の特許請求の範囲の最大の範囲内において権利が保護されるべきであることが理解できよう。

【図面の簡単な説明】

【図１】

メモリ制御論理回路とデータとアドレスレジスタとを備えた従来技術による代表的なフラッシュＥＥＰＲＯＭメモリアレイのブロック図である。

【図２】

システムコントローラを備えた図１のメモリを利用するアーキテクチャを示す。

【図３】

図２のメモリシステムの代表的なコピー処理を示すタイミング図である。

【図４】

複数ページブロックのページのすべてよりも少ない数のページでデータを更新する現行プロセスを示す。

【図５Ａ】

図４の原ブロックに関連する対応する論理ブロックアドレスと物理ブロックアドレスのテーブルである。

【図５Ｂ】

図４の新しいブロックに関連する対応する論理ブロックアドレスと物理ブロックアドレス

複数ページのブロックのページのすべてよりも少ない数のページでデータを更新する別の現行プロセスを示す。

【図７Ａ】

図６の原ブロックに関連する対応する論理ページアドレスと物理ページアドレスのテーブルである。

【図７Ｂ】

図６の新しいブロックに関連する対応する論理ページアドレスと物理ページアドレスのテーブルである。

【図８】

複数ページブロックのページのすべてよりも少ない数のページでデータを更新する改善されたプロセスの一例を示す。

【図９】

図８の新しいブロックに関連する対応する論理的ページ数と物理ページ数のテーブルである。

【図１０】

図８に示すページのデータのレイアウトの一例を示す。

【図１１】

図８の例の別の発展例を示す。

【図１２】

図１１の新しいブロックに関連する対応する論理ページ数と物理ページ数のテーブルである。

【図１３】

更新されたデータを図１１のブロックで読出す１つの方法を示す。

【図１４】

データをプログラムして図８と図９に例示のように組織化されたメモリシステムの中へ入れるプロセスを示すフローチャートである。

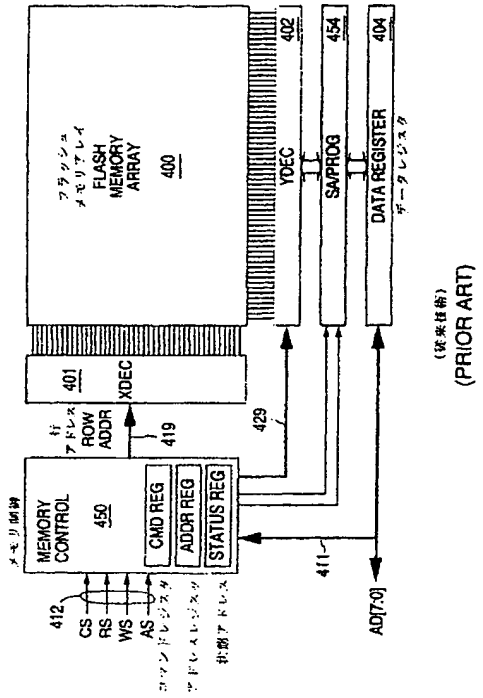
【図１５】

まとめてリンクされて１メタブロックに形成されている個々のユニットから得られるブロックを備えた現行の複数ユニットのメモリを示す。

【図１６】

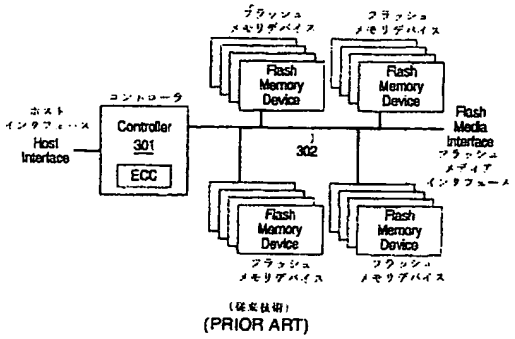
更新されたデータ量がメタブロックのデータ記憶容量よりずっと少ない場合、図１２の複数ユニットメモリ内のメタブロックのデータを更新する改善された方法を示す。

【図１】



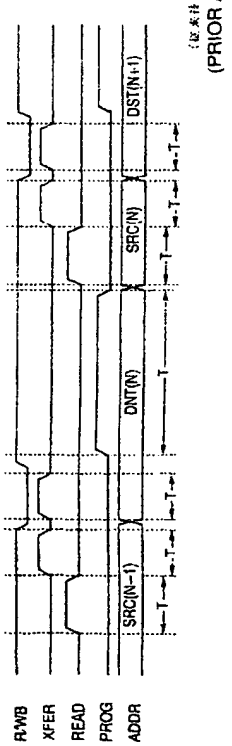
(PRIOR ART)

【図２】



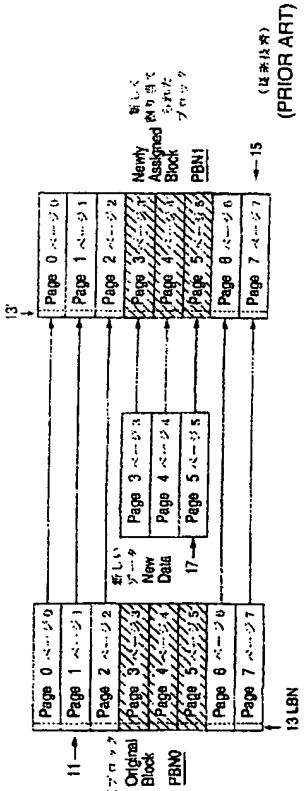
(PRIOR ART)

【図３】



(PRIOR ART)

【図４】



(PRIOR ART)

【図５Ａ】

LBN	PBN
0	0
.....	.....

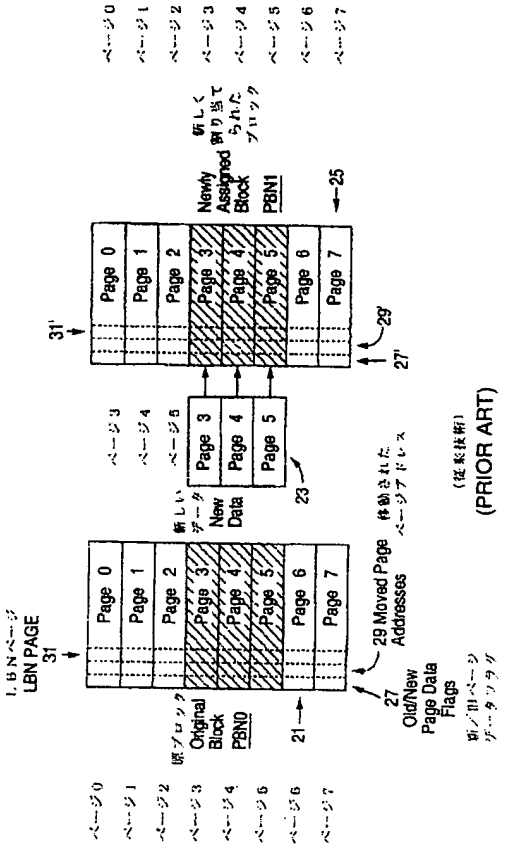
Original Block 11
原ブロック１１

【図５Ｂ】

LBN	PBN
0	1
.....	.....

With New Block 15
新しい
ブロック１５
を具備

【図６】



【図７Ａ】

LBN	Page	PBN	Page
0	0	0	0
0	1	0	1
0	2	0	2
0	3	0	3
0	4	0	4
0	5	0	5
0	6	0	6
0	7	0	7
:	:	:	:

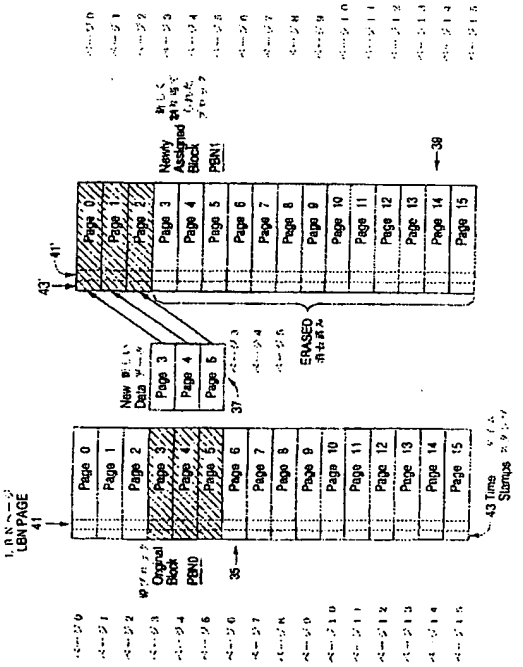
Original Block
原ブロック

【図７Ｂ】

LBN	Page	PBN	Page
0	0	0	0
0	1	0	1
0	2	0	2
0	3	1	3
0	4	1	4
0	5	1	5
0	6	0	6
0	7	0	7
:	:	:	:

With New Block
新しいブロック
を具備

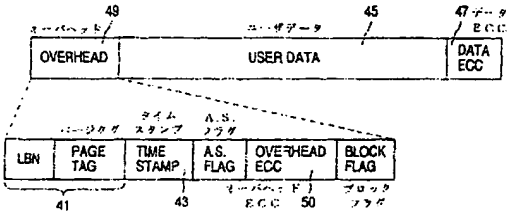
【図８】



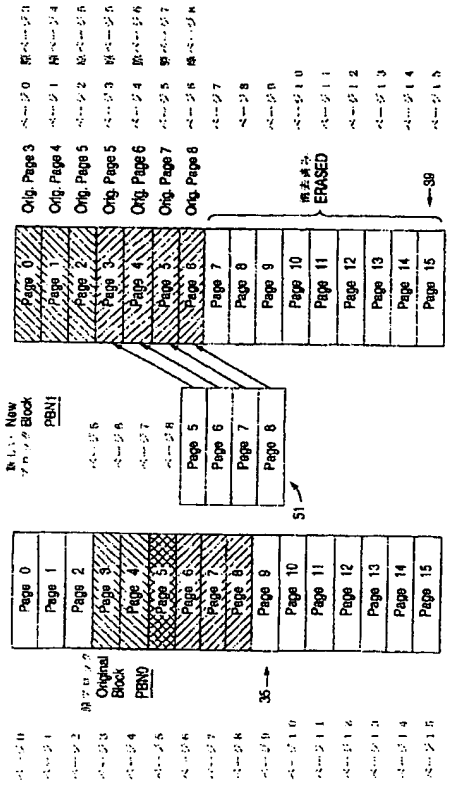
【図９】

LBN	Page	PBN0	Page	PBN1	Page
0	0	0	0		
0	1	0	1		
0	2	0	2		
0	3	0	3	1	0
0	4	0	4	1	1
0	5	0	5	1	2
0	6	0	6		
0	7	0	7		

【図１０】



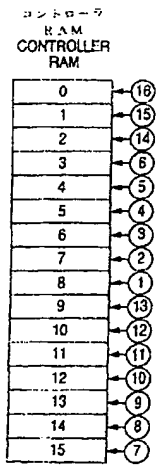
【図１１】



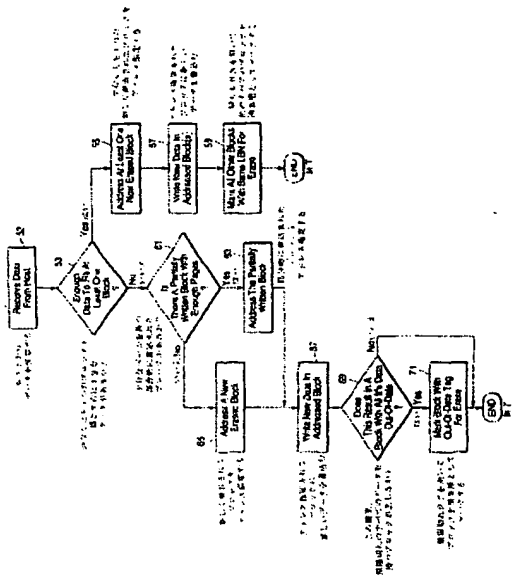
【図１２】

LBN	Page	PBN0	Page	PBN1	Page
0	0	0	0		
0	1	0	1		
0	2	0	2		
0	3	0	3	1	0
0	4	0	4	1	1
0	5	0	5	1	3
0	6	0	6	1	4
0	7	0	7	1	5
0	8	0	8	1	6
0	9	0	9		
:	:	:	:	:	

【図１３】



【図１４】



【国際公開パンフレット】

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
25 July 2002 (25.07.2002)

PCT

(10) International Publication Number
WO 02/058074 A2

(51) International Patent Classification⁷: G11C 16/00

(21) International Application Number: PCT/US02/00066

(22) International Filing Date: 7 January 2002 (07.01.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/765,436 19 January 2001 (19.01.2001) US

(71) Applicant: SANDISK CORPORATION [US/US]; 140 Caspian Court, Sunnyvale, CA 94089 (US).

(72) Inventor: CONLEY, Kevin, M.; 5993 Alvarado Court, San Jose, CA 95120 (US).

(74) Agent: PARSONS, Gerald, P.; Skjerven Morrill MacPherson LLP, Three Embarcadero Ctr. Center, 28th Floor, San Francisco, CA 94111 (US).

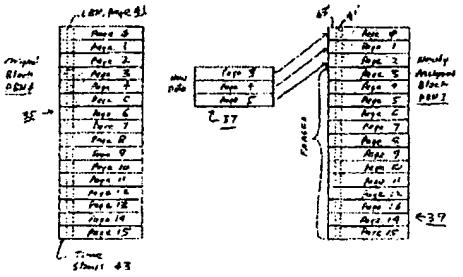
(81) Designated States (national): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States (regional): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— without international search report and to be republished upon receipt of that report

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: PARTIAL BLOCK DATA PROGRAMMING AND READING OPERATIONS IN A NON-VOLATILE MEMORY



(57) Abstract: Data in less than all of the pages of a non-volatile memory block are updated by programming the new data in unused pages of either the same or another block. In order to prevent having to copy unchanged pages of data from the new block, as in program flags into superceded pages of data, the pages of new data are identified by the same logical address as the pages of data which may superceded and a time stamp is added to note when each page was written. When reading the data, the most recent pages of data are used and the older superceded pages of data are ignored. This technique is also applied to metablocks that include one block from each of several different units of a memory array by directing all page writes to a single unused block in one of the units.

WO 02/058074 A2

PARTIAL BLOCK DATA PROGRAMMING AND READING OPERATIONS IN A NON-VOLATILE MEMORY

BACKGROUND OF THE INVENTION

This invention pertains to the field of semiconductor non-volatile data storage system architectures and their methods of operation, and has application to data storage systems based on flash electrically erasable and programmable read-only memories (EEPROMs)

A common application of flash EEPROM devices is as a mass data storage subsystem for electronic devices. Such subsystems are commonly implemented as either removable memory cards that can be inserted into multiple host systems or as non-removable embedded storage within the host system. In both implementations, the subsystem includes one or more flash devices and often a subsystem controller.

Flash EEPROM devices are composed of one or more arrays of transistor cells, each cell capable of non-volatile storage of one or more bits of data. Thus flash memory does not require power to retain the data programmed therein. Once programmed however, a cell must be erased before it can be reprogrammed with a new data value. These arrays of cells are partitioned into groups to provide for efficient implementation of read, program and erase functions. A typical flash memory architecture for mass storage arranges large groups of cells into erasable blocks, wherein a block contains the smallest number of cells (unit of erase) that are erasable at one time.

In one commercial form, each block contains enough cells to store one sector of user data plus some overhead data related to the user data and/or to the block in which it is stored. The amount of user data included in a sector is the standard 512 bytes in one class of such memory systems but can be of some other size. Because the isolation of individual blocks of cells from one another that is required to make them individually erasable takes space on the integrated circuit chip, another class of flash memories makes the blocks significantly larger so there is less space required for such isolation. But since it is also desired to handle user data in much smaller

sectors, each large block is often further partitioned into individually addressable pages that are the basic unit for reading and programming user data (unit of programming and/or reading). Each page usually stores one sector of user data, but a page may store a partial sector or multiple sectors. A "sector" is used herein to refer to an amount of user data that is transferred to and from the host as a unit.

The subsystem controller in a large block system performs a number of functions including the translation between logical addresses (LBAs) received by the memory sub-system from a host, and physical block numbers (PBNs) and page addresses within the memory cell array. This translation often involves use of intermediate terms for a logical block number (LBN) and logical page. The controller also manages the low level flash circuit operation through a series of commands that it issues to the flash memory devices via an interface bus. Another function the controller performs is to maintain the integrity of data stored to the subsystem through various means, such as by using an error correction code (ECC).

In an ideal case, the data in all the pages of a block are usually updated together by writing the updated data to the pages within an unassigned, erased block, and a logical-to-physical block number table is updated with the new address. The original block is then available to be erased. However, it is more typical that the data stored in a number of pages less than all of the pages within a given block must be updated. The data stored in the remaining pages of the given block remains unchanged. The probability of this occurring is higher in systems where the number of sectors of data stored per block is higher. One technique now used to accomplish such a partial block update is to write the data of the pages to be updated into a corresponding number of the pages of an unused erased block and then copy the unchanged pages from the original block into pages of the new block. The original block may then be erased and added to an inventory of unused blocks in which data may later be programmed. Another technique similarly writes the updated pages to a new block but eliminates the need to copy the other pages of data into the new block by changing the flags of the pages in the original block which are being updated to indicate they contain obsolete data. Then when the data are read, the updated data read from pages of the new block are combined with the unchanged data read from pages of the original block that are not flagged as obsolete.

SUMMARY OF THE INVENTION

According to one principal aspect of the present invention, briefly and generally, both the copying of unchanged data from the original to the new blocks and the need to update flags within the original block are avoided when the data of fewer than all of the pages within a block are being updated. This is accomplished by maintaining both the superceded data pages and the updated pages of data with a common logical address. The original and updated pages of data are then distinguished by the relative order in which they were programmed. During reading, the most recent data stored in the pages having the same logical address are combined with the unchanged pages of data while data in the original versions of the updated pages are ignored. The updated data can be written to either pages within a different block than the original data, or to available unused pages within the same block. In one specific implementation, a form of time stamp is stored with each page of data that allows determining the relative order that pages with the same logical address were written. In another specific implementation, in a system where pages are programmed in a particular order within the blocks, a form of time stamp is stored with each block of data, and the most recent copy of a page within a block is established by its physical location within the block.

These techniques avoid both the necessity for copying unchanged data from the original to new block and the need to change a flag or other data in the pages of the original block whose data have been updated. By not having to change a flag or other data in the superceded pages, a potential of disturbing the previously written data in adjacent pages of that same block that can occur from such a writing operation is eliminated. Also, a performance penalty of the additional program operation is avoided.

A further operational feature, which may be used in conjunction with the above summarized techniques, keeps track of the logical offset of individual pages of data within the individual memory cell blocks, so that the updated data need not be stored with the same physical page offset as the superceded data. This allows more efficient use of the pages of new blocks, and even allows the updated data to be stored in any erased pages of the same block as the superceded data.

Another principal aspect of the present invention groups together two or more blocks positioned in separate units of the memory array (also termed "sub-arrays") for

programming and reading together as part of a single operation. Such a multiple block group is referenced herein as a "metablock." Its component blocks may be either all located on a single memory integrated circuit chip, or, in systems using more than one such chip, located on two or more different chips. When data in fewer than all of the pages of one of these blocks is updated, the use of another block in that same unit is normally required. Indeed, the techniques described above, or others, may be employed separately with each block of the metablock. Therefore, when data within pages of more than one block of the metablock are updated, pages within more than one additional block are required to be used. If there are four blocks of four different memory units that form the metablock, for example, there is some probability that up to an additional four blocks, one in each of the units, will be used to store updated pages of the original blocks. One update block is potentially required in each unit for each block of the original metablock. In addition, according to the present invention, updated data from pages of more than one of the blocks in the metablock can be stored in pages of a common block in only one of the units. This significantly reduces the number of unused erased blocks that are needed to store updated data, thereby making more efficient use of the available memory cell blocks to store data. This technique is particularly useful when the memory system frequently updates single pages from a metablock.

Additional aspects, features and advantages of the present invention are included in the following description of exemplary embodiments, which description should be read in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram of a typical prior art flash EEPROM memory array with memory control logic, data and address registers;

Figure 2 illustrates an architecture utilizing memories of Figure 1 with a system controller;

Figure 3 is a timing diagram showing a typical copy operation of the memory system of Figure 2;

Figure 4 illustrates an existing process of updating data in less than all of the pages of a multi-paged block;

Figures 5A and 5B are tables of corresponding logical and physical block addresses for each of the original and new blocks of Figure 4, respectively;

Figure 6 illustrates another existing process of updating data in less than all of the pages of a multi-paged block;

Figures 7A and 7B are tables of corresponding logical and physical page addresses for the original and new blocks of Figure 6, respectively;

Figure 8 illustrates an example of an improved process of updating data in less than all of the pages of a multi-paged block;

Figure 9 is a table of corresponding logical and physical page numbers for the new block of Figure 8;

Figure 10 provides an example of a layout of the data in a page shown in Figure 8;

Figure 11 illustrates a further development of the example of Figure 8;

Figure 12 is a table of corresponding logical and physical page numbers for the new block of Figure 11;

Figure 13 illustrates one way to read the updated data in the blocks of Figure 11;

Figure 14 is a flow diagram of a process of programming data into a memory system organized as illustrated in Figures 8 and 9; Figure 15 illustrates an existing multi-unit memory with blocks from the individual units being linked together into a metablock and

Figure 16 illustrates an improved method of updating data of a metablock in the multi-unit memory of Figure 12 when the amount of updated data is much less that the data storage capacity of the metablock.

DESCRIPTION OF EXISTING LARGE BLOCK MANAGEMENT TECHNIQUES

Figure 1 shows a typical flash memory device internal architecture. The primary features include an input/output (I/O) bus 411 and control signals 412 to interface to an external controller, a memory control circuit 450 to control internal memory operations with registers for command, address and status signals. One or more arrays 400 of flash EEPROM cells are included, each array having its own row decoder (XDEC) 401 and column decoder (YDEC) 402, a group of sense amplifiers

and program control circuitry (SA/PROG) 454 and a data register 404. Presently, the memory cells usually include one or more conductive floating gates as storage elements but other long term electron charge storage elements may be used instead. The memory cell array may be operated with two levels of charge defined for each storage element to therefore store one bit of data with each element. Alternatively, more than two storage states may be defined for each storage element, in which case more than one bit of data is stored in each element.

If desired, a plurality of arrays 400, together with related X decoders, Y decoders, program/verified circuitry, data registers, and the like are provided, for example as taught by U.S. Patent 5,890,192, issued March 30, 1999, and assigned to Sandisk Corporation, the assignee of this application, which is hereby incorporated by this reference. Related memory system features are described in co-pending patent application serial no. 09/505,555, filed February 17, 2000 by Kevin Conley et al., which application is expressly incorporated herein by this reference.

The external interface I/O bus 411 and control signals 412 can include the following:

CS - Chip Select.	Used to activate flash memory interface.
RS - Read Strobe.	Used to indicate the I/O bus is being used to transfer data from the memory array.
WS - Write Strobe.	Used to indicate the I/O bus is being used to transfer data to the memory array.
AS - Address Strobe.	Indicates that the I/O bus is being used to transfer address information.
AD[7:0] - Address/Data Bus	This I/O bus is used to transfer data between controller and the flash memory command, address and data registers of the memory control 450.

This interface is given only as an example as other signal configurations can be used to give the same functionality. Figure 1 shows only one flash memory array 400 with its related components, but a multiplicity of such arrays can exist on a single flash memory chip that share a common interface and memory control circuitry but have separate XDEC, YDEC, SA/PROG and DATA REG circuitry in order to allow parallel read and program operations.

Data is transferred from the memory array through the data register 404 to an external controller via the data registers' coupling to the I/O bus AD[7:0] 411. The data register 404 is also coupled the sense amplifier/programming circuit 454. The number of elements of the data register coupled to each sense amplifier/programming circuit element may depend on the number of bits stored in each storage element of the memory cells, flash EEPROM cells each containing one or more floating gates as the storage elements. Each storage element may store a plurality of bits, such as 2 or 4, if the memory cells are operated in a multi-state mode. Alternatively, the memory cells may be operated in a binary mode to store one bit of data per storage element.

The row decoder 401 decodes row addresses for the array 400 in order to select the physical page to be accessed. The row decoder 401 receives row addresses via internal row address lines 419 from the memory control logic 450. A column decoder 402 receives column addresses via internal column address lines 429 from the memory control logic 450.

Figure 2 shows an architecture of a typical non-volatile data storage system, in this case employing flash memory cells as the storage media. In one form, this system is encapsulated within a removable card having an electrical connector extending along one side to provide the host interface when inserted into a receptacle of a host. Alternatively, the system of Figure 2 may be embedded into a host system in the form of a permanently installed embedded circuit or otherwise. The system utilizes a single controller 301 that performs high level host and memory control functions. The flash memory media is composed of one or more flash memory devices, each such device often formed on its own integrated circuit chip. The system controller and the flash memory are connected by a bus 302 that allows the controller 301 to load command, address, and transfer data to and from the flash memory array. The controller 301 interfaces with a host system (not shown) with which user data is transferred to and from the flash memory array. In the case where the system of Figure 2 is included in a card, the host interface includes a mating plug and socket assembly (not shown) on the card and host equipment.

The controller 301 receives a command from the host to read or write one or more sectors of user data starting at a particular logical address. This address may or may not align with a boundary of a physical block of memory cells.

In some prior art systems having large capacity memory cell blocks that are divided into multiple pages, as discussed above, the data from a block that is not being updated needs to be copied from the original block to a new block that also contains the new, updated data being written by the host. This technique is illustrated in Figure 4, wherein two of a large number of blocks of memory are included. One block 11 (PBN0) is illustrated to be divided into 8 pages for storing one sector of user data in each of its pages. Overhead data fields contained within each page include a field 13 containing the LBN of the block 11. The order of the logical pages within a logical block is fixed with respect to the corresponding physical pages within a physical block. A second similarly configured block 15 (PBN1) is selected from an inventory of unused, erased blocks. Data within pages 3-5 of the original block 11 are being updated by three pages of new data 17. The new data is written into the corresponding pages 3-5 of the new block 15, and user data from pages 0-2, 6 and 7 of the block 11 are copied into corresponding pages of the new block 15. All pages of the new block 15 are preferably programmed in a single sequence of programming operations. After the block 15 is programmed, the original block 11 can be erased and placed in inventory for later use. The copying of data between the blocks 11 and 15, which involves reading the data from one or more pages in the original block and subsequently programming the same data to pages in a newly assigned block, greatly reduces the write performance and usable lifetime of the storage system.

With reference to Figures 5A and 5B, partial tables show mapping of the logical blocks into the original and new physical blocks 11 and 15 before (Figure 5A) and after (Figure 5B) the updating of data described with respect to Figure 4. Before the data update, the original block 11, in this example, stores pages 0-7 of LBN0 into corresponding pages 0-7 of PBN0. After the data update, the new block 15 stores pages 0-7 of LBN0 in corresponding pages 0-7 of PBN1. Receipt of a request to read data from LBN0 is then directed to the physical block 15 instead of the physical block 11. In a typical controller operation, a table in the form of that shown in Figures 5A and 5B is built from the LBN field 13 read from a physical page and knowledge of the PBN that is addressed when reading the data field 13. The table is usually stored in a volatile memory of the controller for ease of access, although only a portion of a complete table for the entire system is typically stored at any one time. A portion of

the table is usually formed immediately in advance of a read or programming operation that involves the blocks included in the table portion.

In other prior art systems, flags are recorded with the user data in pages and are used to indicate that pages of data in the original block that are being superceded by the newly written data are invalid. Only the new data is written to the newly assigned block. Thus, the data in pages of the block not involved in the write operation but contained in the same physical block as the superceded data need not be copied into the new block. This operation is illustrated in Figure 6, where pages 3-5 of data within an original block 21 (PBN0) are again being updated. Updated pages 3-5 of data 23 are written into corresponding pages of a new block 25. As part of the same operation, an old/new flag 27 is written in each of the pages 3-5 to indicate the data of those pages is old, while the flag 27 for the remaining pages 0-2, 6 and 7 remains set at "new". Similarly, the new PBN1 is written into another overhead data field of each of the pages 3-5 in the block 21 to indicate where the updated data are located. The LBN and page are stored in a field 31 within each of the physical pages.

Figures 7A and 7B are tables of the correspondence between the data LBN/page and the PBN/page before (Figure 7A) and after (Figure 7B) the data update is complete. The unchanged pages 0-2, 6 and 7 of the LBN remain mapped into PBN0 while the updated pages 3-5 are shown to reside in PBN1. The table of Figure 7B is built by the memory controller by reading the overhead data fields 27, 29 and 31 of the pages within the block PBN0 after the data update. Since the flag 27 is set to "old" in each of pages 3-5 of the original block PBN0, that block will no longer appear in the table for those pages. Rather, the new block number PBN1 appears instead, having been read from the overhead fields 29' of the updated pages. When data are being read from LBN0, the user data stored in the pages listed in the right column of Figure 7B are read and then assembled in the order shown for transfer to the host.

Various flags are typically located in the same physical page as the other associated overhead data, such as the LBN and an ECC. Thus, to program the old/new flags 27, and others, in pages where the data has been superceded requires that a page support multiple programming cycles. That is, the memory array must have the capability that its pages can be programmed in at least at least two stages between erasures. Furthermore, the block must support the ability to program a page

when other pages in the block with higher offsets or addresses have been already programmed. A limitation of some flash memories however prevents the usage of such flags by specifying that the pages in a block can only be programmed in a physically sequential manner. Furthermore, the pages support a finite number of program cycles and in some cases additional programming of programmed pages is not permitted.

What is needed is a mechanism by which data that partially supercedes data stored in an existing block can be written without either copying unchanged data from the existing block or programming flags to pages that have been previously programmed.

DESCRIPTION OF EXEMPLARY EMBODIMENTS OF THE INVENTION

There are many different types of flash EEPROM, each of which presents its own limitations that must be worked around to operate a high performance memory system formed on a small amount of integrated circuit area. Some do not provide for writing any data into a page that has already been programmed, so updating flags in a page that contains superceded data, as described above, is not possible. Others allow such flags to be written but doing so in pages whose data is being superceded can disturb data in other pages of the same block that remain current.

An example memory system where this has been found to be a problem is a NAND type, where a column of memory cells is formed as a series circuit string between a bit line and a common potential. Each word line extends across a row of memory cells formed of one cell in each such string. Such a memory is particularly susceptible to such memory state disturbs when being operated in a multi-state mode to store more than one bit of data in each such cell. Such operation divides an available window of a memory cell transistor threshold voltage range into narrow non-overlapping voltage level ranges, each range becoming narrower as the number of levels, and thus the number of bits being stored in each cell, are increased. For example, if four threshold ranges are used, two bits of data are stored in each cell's storage element. And since each of the four threshold voltage ranges is necessarily small, the chance of the state of a cell being disturbed by programming other cells in the same block is increased with multi-state operation. In this case, the writing of the

old/new or other flags, as described with respect to Figures 6, 7A and 7B, cannot be tolerated.

A common feature of each of the existing memory management techniques described above with respect to Figures 4-7B is that a logical block number (LBN) and page offset is mapped within the system to at most two physical block numbers (PBNs). One block is the original block and the other contains the updated page data. Data are written to the page location in the block corresponding to the low order bits of its logical address (LBA). This mapping is typical in various types of memory systems. In the techniques described below, pages containing updated data are also assigned the same LBN and page offsets as the pages whose data has been superceded. But rather than tagging the pages containing original data as being superceded, the memory controller distinguishes the pages containing the superceded data from those containing the new, updated version either (1) by keeping track of the order in which the pages having the same logical addresses were written, such as by use of a counter, and/or (2) from the physical page addresses wherein, when pages are written in order within blocks from the lowest page address to the highest, the higher physical address contains the most recent copy of the data. When the data is accessed for reading, therefore, those in the most current pages are used in cases where there are pages containing superceded data that have the same logical addresses, while the superceded data are ignored.

A first specific implementation of this technique is described with respect to Figures 8 and 9. The situation is the same in this example as that in the prior art techniques described with respect to Figures 4-7B, namely the partial re-write of data within a block 35, although each block is now shown to contain 16 pages. New data 37 for each of the pages 3-5 of the block 35 (PBN 35) is written into three pages of a new block 39 (PBN1) that has previously been erased, similar to that described previously. A LBN and page offset overhead data field 41 written into the pages of PBN1 that contain the updated data is the same as that in the pages of the superceded data in the initial block PBN0. The table of Figure 9, formed from the data within the fields 41 and 41', shows this. The logical LBN and page offsets, in the first column, are mapped into both the first physical block (PBN0), in the second column, and, for the pages that have been updated, also into the second physical block (PBN1) in the third column. The LBN and logical page offsets 41' written into each of the three

pages of updated data within the new block PBN1 are the same as those 41 written into each of a corresponding logical page of the original block PBN0.

In order to determine which of two pages having the same LBN and page offset contains the updated data, each page contains another overhead field 43 that provides an indication of its time of programming, at least relative to the time that other pages with the same logical address are programmed. This allows the controller to determine, when reading the data from the memory, the relative ages of the pages of data that are assigned the same logical address.

There are several ways in which the field 43, which contains a form of time stamp, may be written. The most straight forward way is to record in that field, when the data of its associated page is programmed, the output of a real-time clock in the system. Later programmed pages with the same logical address then have a later time recorded in the field 43. But when such a real-time clock is not available in the system, other techniques can be used. One specific technique is to store the output of a modulo-N counter as the value of the field 43. The range of the counter should be one more than the number of pages that are contemplated to be stored with the same logical page number. When updating the data of a particular page in the original block PBN0, for example, the controller first reads the count stored in the field 43 of the page whose data are being updated, increments the count by some amount, such as one, and then writes that incremented count in the new block PBN1 as the field 43'. The counter, upon reaching a count of N+1, rolls over to 0. Since the number of blocks with the same LBN is less than N, there is always a point of discontinuity in the values of stored counts. It is easy then to handle the rollover with normalized to the point of discontinuity.

The controller, when called upon to read the data, easily distinguishes between the new and superceded pages' data by comparing the counts in the fields 43 and 43' of pages having the same LBA and page offset. In response to a need to read the most recent version of a data file, data from the identified new pages are then assembled, along with original pages that have not been updated, into the most recent version of the data file.

It will be noted that, in the example of Figure 8, the new data pages 37 are stored in the first three pages 0-2 of the new block PBN1, rather than in the same pages 3-5 which they replace in the original block PBN0. By keeping track of the

individual logical page numbers, the updated data need not necessarily be stored in the same page offset of the new block as that of the old block where superceded data is contained. Page(s) of updated data can also be written to erased pages of the same block as the page of data being superceded.

As a result, there is no constraint presented by the techniques being described that limit which physical page new data can be written into. But the memory system in which these techniques are implemented may present some constraints. For example, one NAND system requires that the pages within the blocks be programmed in sequential order. That means that programming of the middle pages 3-5, as done in the new block 25 (Figure 6), wastes the pages 0-2, which cannot later be programmed. By storing the new data 37 in the first available pages of the new block 39 (Figure 8) in such a restrictive system, the remaining pages 3-7 are available for later use to store other data. Indeed, if the block 39 had other data stored in its pages 0-4 at the time the three pages of new data 37 were being stored, the new data could be stored in the remaining unused pages 5-7. This makes maximum use of the available storage capacity for such a system.

An example of the structure of data stored in an individual page of the blocks of Figure 8 is shown in Figure 10. The largest part is user data 45. An error correction code (ECC) 47 calculated from the user data is also stored in the page. Overhead data 49, including the LBN and page tag 41 (logical page offset), the time stamp 43 and an ECC 51 calculated from the overhead data are also stored in the page. By having an ECC 50 covering the overhead data that is separate from the user data ECC 47, the overhead 49 may be read separately from the user data and evaluated as valid without the need to transfer all of the data stored in the page. Alternatively, however, where the separate reading of the overhead data 49 is not a frequent event, all of the data in the page may be covered by a single ECC in order to reduce the total number of bits of ECC in a page.

A second specific implementation of the inventive technique can also be described with respect to Figure 8. In this example, the time stamp is used only to determine the relative age of the data stored in blocks, while the most recent pages among those that carry the same LBN and page number are determined by their relative physical locations. The time stamp 43 then does not need to be stored as part of each page. Rather, a single time stamp can be recorded for each block, either as

part of the block or elsewhere within the non-volatile memory, and is updated each time a page of data is written into the block. Data is then read from pages in an order of descending physical address, starting from the last page of the most recently updated block containing data pages having the same LBN.

In Figure 8, for example, the pages are first read in the new block PBN1 from the last (page 15) to the first (page 0), followed by reading the pages of the original block PBN0 in the same reverse order. Once logical pages 3, 4 and 5 have been read from the new block PBN1, the superceded data in those pages of the original block PBN0 that are identified by the same logical page numbers can be skipped during the reading process. Specifically, physical pages 3, 4 and 5 of the old block PBN0 are skipped during reading, in this example, once the controller determines that their LBN/pages 41 are the same as those of the pages already read from the new block PBN1. This process can increase the speed of reading and reduce the number of overhead bits 49 that need to be stored for each page. Further, when this reverse page reading technique is employed, the table of Figure 9 used by the controller during a reading operation can be simplified into the form of Figures 5A and 5B. Only an identity of those physical blocks containing data of a common logical block and the relative times that the physical blocks were programmed need to be known in order to carry out this efficient reading process.

Figure 11 illustrates an extension of the example of Figure 8 by including a second update to the data originally written in the block PBN0. New data 51 for logical pages 5, 6, 7 and 8 is written to the respective physical pages 3, 4, 5 and 6 of the new block PBN1, along with their LBN and page number. Note, in this example, that the data of logical page 5 is being updated for the second time. During a reading operation that begins from the last page of the new block PBN1, the most recently written logical pages 8, 7, 6 and 5 of the data of interest are first read in that order. Thereafter, it will be noted that the LBN/page overhead field in physical page 2 of PBN1 is the same as that read from the physical page 3, so the user data of page 2 is not read. The physical pages 1 and 0 are then read. Next, the pages of the original block PBN0 are read, beginning with physical page 15. After reading physical pages 15-9, the controller will note that the LBN/page fields of each of pages 8-3 match those of pages whose data has already been read, so the old data need not be read

from those pages. The efficiency of the reading process is thus improved. Finally, the original data of physical pages 2-0 are read since that data was not updated.

It will be noted that this example of reading pages in a reverse order efficiently sorts out the new data pages from the superceded data pages because data are written in physical page locations of an erased block in order from page 0 on. This technique is not limited to use with a memory system having such a specific programming constraint, however. So long as the order in which pages are programmed within a given block is known, the data from those pages may be read in the reverse order from which they were written. What is desired is that the most recently programmed pages having a common LBN with others that were earlier programmed be read first, and these are the most recently programmed pages. The most recent versions of updated pages are read first so that the superceded versions may easily be identified thereafter.

A table showing the correspondence between the logical data and physical page addresses for the example of Figure 11 is given in Figure 12. Although there have been two data updates, both are represented by the single column for the second block PBN1. The physical page noted in PBN1 for the logical page 5 is simply changed upon the second update to that page occurring. If the updating involves a third block, then another column is added for that other block. The table of Figure 12, constructed by reading the overhead data from each of the pages in blocks to which data of a common LBN has been written, can be used by the first implementation when the reverse page reading technique is not used. When the reverse page reading technique described above is used, the table of Figure 12 need be built only to identify a correspondence between an LBN and all PBNs containing data of that LBN.

An efficient way to organize pages of data being read from a physical block, where one or more of the pages has been updated, is illustrated by Figure 13. Enough space is provided in a volatile memory of the controller to buffer at least several pages of data at a time, and preferably a full block of data. That is what is shown in Figure 13. Sixteen pages of data, equal to the amount stored in a non-volatile memory block, are stored in the controller memory. Since the pages are most commonly read out of order, each page of data is stored in its proper position with respect to the other pages. For example, in the reverse page read operation of Figure 11, logical page 8 if the first to be read, so it is stored in position 8 of the controller memory, as indicated by the

"1" in a circle. The next is logical page 7, and so forth, until all pages of data desired by the host are read and stored in the controller memory. The entire set of page data is then transferred to the host without having to manipulate the order of the data in the buffer memory. The pages of data have already be organized by writing them to the proper location in the controller memory.

A method of programming a non-volatile memory system that utilizes the techniques described with respect to Figures 8 and 9 is illustrated in the flow chart of Figure 14. Data for pages of an existing file to be updated are received from a host system, as indicated by the block 52. It is first determined by a step 53 whether the number of pages of updated data to be stored is equal to or greater than the storage capacity of a block of the system, 16 pages being shown as the block capacity, for simplicity, in the above described example. If so, one or more unused, erased blocks are addressed, in a step 55, and the new data pages are written to the addressed block(s), in a step 57. Typically, the updating of one block or more of data will result in one or more blocks storing the data that have been superceded by the new data. If so, as indicated by a step 59, those blocks with superceded data are identified for erasure. For the purpose of increasing performance, it is preferable that erase operations occur in the background, or when host requested programming or reading operations are not taking place. After being erased, the blocks are returned to the inventory of unused, erased blocks for further use. Alternatively, erasure of the blocks can be deferred until they are needed for programming operations.

If, on the other hand, in the step 53, it is determined that there are fewer pages of new data than will utilize the full storage capacity of a block, a next step 61 determines whether there are enough unused pages in a block having some pages programmed with other data. If so, such a block is addressed, in a step 63. If not, a totally unused, erased block is addressed, in a step 65. In either case, in a step 67, the new data are programmed into unused pages of the addressed block. As part of this programming process, the LBN and page offset is written into the fields 41, and the time stamp into the fields 43 of each of the pages (Figure 8) of the updated data, in the manner described above.

A desirable feature of the programming process is to make available for future programming any blocks that store only superceded data. So the question is asked, in a step 69, whether the data updating process has resulted in an entire block remaining

with only superceded data. If so, such a block is queued for erasure, in a step 71, and the process is then completed. If not, the step 71 is omitted and the data update is finished.

METABLOCK OPERATION

In order to improve performance by reducing programming time, a goal is to program as many cells in parallel as can reasonably be done without incurring other penalties. One implementation divides the memory array into largely independent sub-arrays or units, such as multiple units 80-83 of Figure 15, each unit in turn being divided into a large number of blocks, as shown. Pages of data are then programmed at the same time into more than one of the units. Another configuration further combines one or more of these units from multiple memory chips. These multiple chips may be connected to a single bus (as shown in Figure 2) or multiple independent busses for higher data throughput. An extension of this is to link blocks from different units for programming, reading and erasing together, an example being shown in Figure 15. Blocks 85-88 from respective ones of the units 80-83 can be operated together as a metablock, for example. As with the memory embodiments described above, each block, the smallest erasable group of the memory array, is typically divided into multiple pages, a page containing the smallest number of cells that are programmable together within the block. Therefore, a programming operation of the metablock shown in Figure 15 will usually include the simultaneously programming of data into at least one page of each of the blocks 85-88 forming the metablock, which is repeated until the metablock is full or the incoming data has all been programmed. Other metablocks are formed of different blocks from the array units, one block from each unit.

In the course of operating such a memory, as with others, pages of data less than an entire block often need to be updated. This can be done for individual blocks of a metablock in the same manner as described above with respect to either of Figures 4 or 6, but preferably by use of the improved technique described with respect to Figure 8. When any of these three techniques are used to update data of one block of the metablock, an additional block of memory within the same unit is also used. Further, a data update may require writing new data for one or more pages of two or more of the blocks of a metablock. This can then require use of up to four additional

blocks 90-93, one in each of the four units, to update a data file stored in the metablock, even though the data in only a few pages is being updated.

In order to reduce the number of blocks required for such partial block updates, according to another aspect of the present invention, updates to pages of data within any of the blocks of the illustrated metablock are made, as illustrated by Figure 16, to a single additional block 90 in the memory unit 80, so long as unused pages in the block 80 remain. If, for example, data in three pages of the block 86 and two pages of the block 88 are being updated at one time, all five pages of the new data are written into the block 90. This can save the use of one block of memory, thereby to effectively increase the number of available erased blocks by one block. This helps avoid, or at least postpone, the time when an inventory of erased blocks becomes exhausted. If one or more pages from each of the four blocks 85-88 are being updated, all of the new data pages are programmed in the single block 90, thereby avoiding tying up an additional three blocks of memory to make the update. If the number of pages of new data exceed the capacity of an unused block, pages that the block 90 cannot accept are written to another unused block which may be in the same unit 80 or one of the other units 81-83.

Although the invention has been described with respect to various exemplary embodiments, it will be understood that the invention is entitled to protection within the full scope of the appended claims.

IT IS CLAIMED:

1. A method of simultaneously storing original and replacement data in a non-volatile memory system, comprising:

identifying the original and replacement data by the same logical address, and

distinguishing the replacement data from the original data by keeping track of the relative times that the original and replacement data have been programmed into the memory.

2. A method of storing and retrieving original and replacement data in a non-volatile memory system, comprising:

identifying units of the original and the replacement data by the same logical address,

reading units of data in an inverse order from an order in which they were programmed into the memory, and

distinguishing units of replacement data from units of original data having the same logical address by the order in which they are read.

3. In a non-volatile memory system having a plurality of blocks of memory storage elements that are individually organized into a plurality of pages of memory storage elements, a method of substituting new data for superceded data within at least one page of one of the plurality of blocks while data in at least another page of said one block is not replaced, comprising:

programming the new data into at least one page of said one or another of the plurality of blocks,

identifying the at least one page of superceded data and the at least one page of new data by a common logical address, and

recording a relative time of programming the new and the superceded data.

4. The method of claim 3, wherein the relative time of programming is recorded for the individual pages in which the new and superceded data are programmed, whereby the at least one page of new data is distinguishable from the at least one page of superceded data by their recorded relative times of programming.

5. The method of claim 3, wherein the relative time of programming is recorded for the individual blocks, thereby to identify an order of programming of individual blocks having data with a common logical address, and further wherein pages within the individual blocks are programmed in a designated order, whereby the new pages of data are distinguishable from superceded pages of data within an individual block by their relative positions within the block.

6. The method of claim 3, wherein the data in at least another page of said one block that is not replaced are not copied into said one or another block as part of substituting the new data for the superceded data.

7. The method of claim 3, wherein nothing is written into the at least one page of superceded data as part of substituting the new data for the superceded data.

8. The method of claim 4, wherein recording a relative time of programming the new and superceded data includes storing a value of a clock at each of the times that the new and superceded data are programmed.

9. The method of claim 4, wherein recording a relative time of programming the new and superceded data includes storing a different value of a sequence of numbers at each of the times that the new and superceded data are programmed.

10. The method of either of claims 8 or 9, wherein storing the value indicating a relative time of programming the new and superceded data includes storing the individual values within the same pages as the new and superceded data to which the values relate.

11. The method of claim 3, wherein programming the new data into at least one page of another said one or another of the plurality of blocks includes programming the new data into the first available unused pages within said one or another block in a predefined order

12. The method of claim 3, wherein identifying the at least one page of superceded data and the at least one page of new data by a common logical address includes recording at least part of the common logical address in the individual pages as overhead data.

13. The method of claim 12, including building a table in volatile memory including multiple physical block addresses for the common logical address.

14. A method of reading data that has been updated according to claim 4, comprising:

reading pages of data from said one block and, if new data has been programmed thereinto, said another block,

identifying any multiple pages of data that have the same logical address,

utilizing the recorded relative time of programming the new and superceded data to identify the most current of any pages having the same logical address, and

assembling data in the most current of any pages having the same logical address along with pages in said at least another page of said one block that have not been updated.

15. A method of reading data that has been updated according to claim 5, comprising:

reading pages of data within said one and, if new data has been programmed thereinto, another block in a reverse order from which they were programmed, and

passing over any pages of data so read which have the same logical page address as a page whose data has already been read.

16. The method of either one of claims 14 or 15, additionally comprising operating the individual memory storage elements with more than two storage states, thereby storing more than one bit of data in each storage element, and reading pages of data includes reading the more than two storage states from the individual memory storage elements.

17. The method of any one of claims 3-9, additionally comprising operating storage elements of the individual memory cells with more than two storage states, thereby storing more than one bit of data in each storage element.

18. The method of claim 17, wherein the storage elements include individual floating gates.

19. The method of any one of claims 3-9, wherein the non-volatile memory system is formed within an enclosed card having an electrical connector along one edge thereof that operably connects with a host system.

20. A method of operating a non-volatile memory system having an array of memory storage elements organized into at least two sub-arrays, wherein the individual sub-arrays are divided into a plurality of non-overlapping blocks of storage elements wherein a block contains the smallest group of memory storage elements that are erasable together, and the individual blocks are divided into a plurality of pages of storage elements wherein a page is the smallest group of memory storage elements that are programmable together, comprising:

linking at least one block from individual ones of said at least two sub-arrays to form a metablock wherein its component blocks are erased together as a unit, and

updating pages of original data within any of the metablock component blocks less than all the pages within the block by programming replacement data into pages within another at least one block in only a designated one of the sub-arrays regardless of which sub-array the data being updated is stored.

21. The method of claim 20, wherein storing the original and replacement data includes:

identifying the original and replacement data by the same logical address to the memory system, and

distinguishing the replacement data from the original data by keeping track of the relative times that the original and replacement data have been programmed their respective pages of the memory.

22. A non-volatile memory system, comprising:

an array of non-volatile memory storage elements organized in blocks of storage elements, wherein an individual block contains the smallest group of storage elements that is erasable, and

a programming mechanism that writes into a first block an updated version of less than all of original data stored in a second block along with an indication of the later writing of the updated version,

an address mechanism that logically addresses both the original data and the updated version with the same address, and

a reading mechanism that distinguishes the updated version from the original data at least in part by the relative time by said indication of the later writing of the updated version.

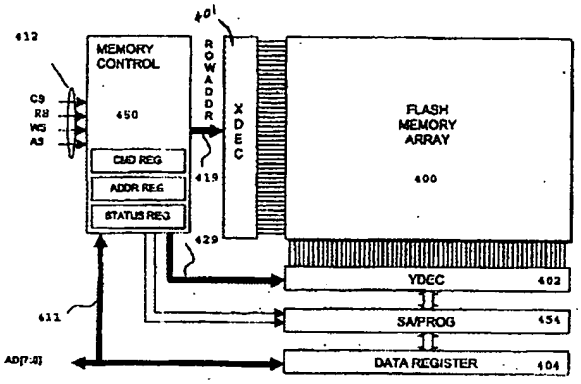


Fig. 1 (Prior Art)

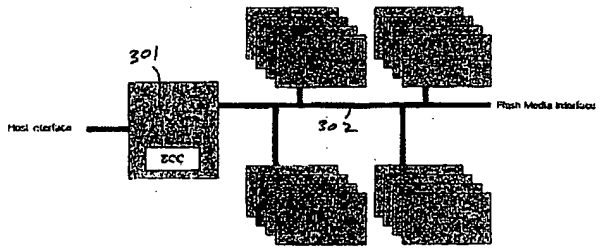


Fig. 2 (Prior Art)

JP 2004-533029 A 2004.10.28

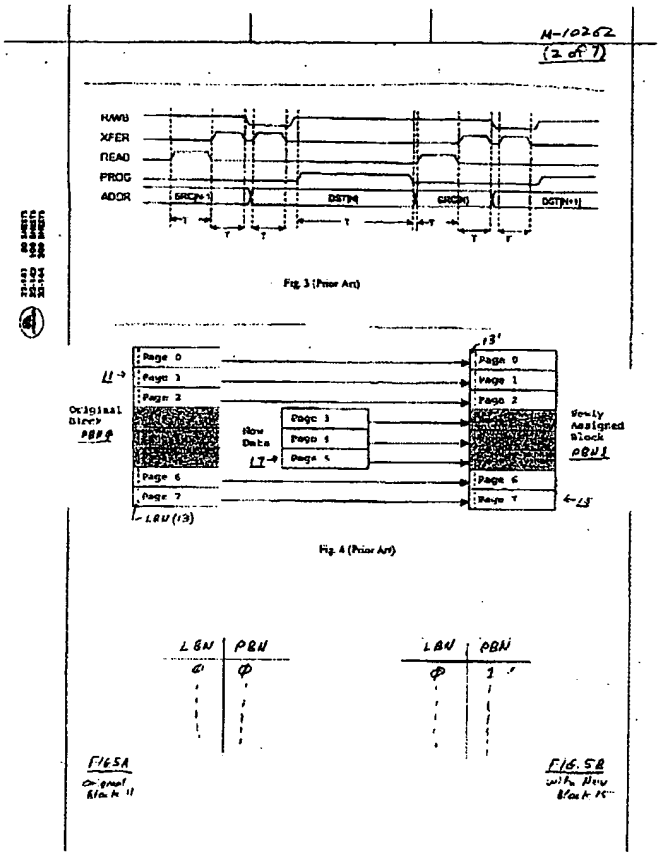
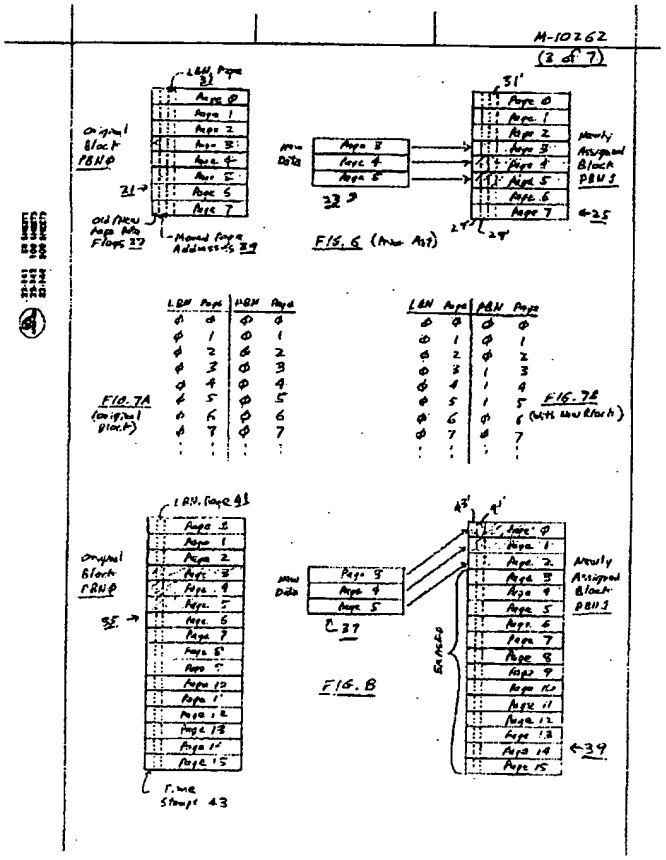
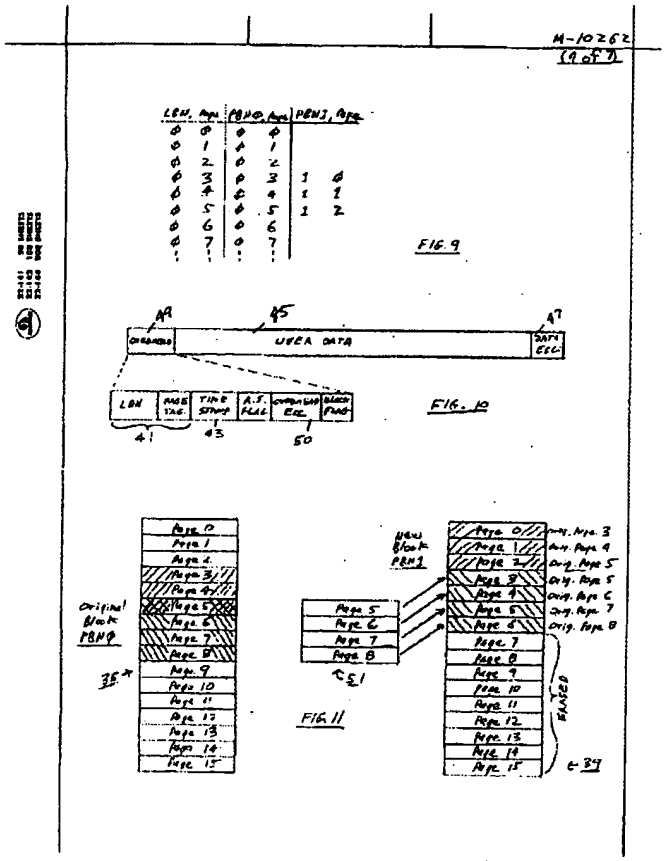


Fig. 3 (Prior Art)

Fig. 4 (Prior Art)


M-10262
(3 of 7)
Original Block PBN#
Newly Assigned Block PBN#
FIG. 6
FIG. 7A
FIG. 7B
FIG. 8
Expand

JP 2004-533029 A 2004.10.28



JP 2004-533029 A 2004.10.28

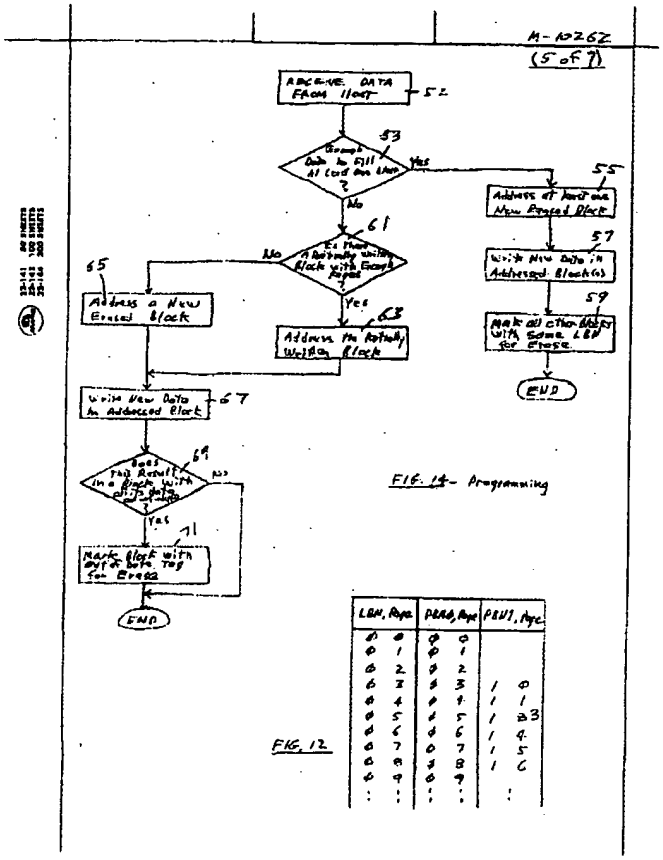


FIG. 14- Programming

FIG. 12

LBN, Byte		PBA, Byte		PBVI, Byte	
0	0	0	0		
0	1	0	1		
0	2	0	2		
0	3	0	3	1	0
0	4	0	4	1	1
0	5	0	5	1	B3
0	6	0	6	1	4
0	7	0	7	1	5
0	8	0	8	1	C
0	9	0	9		
:	:	:	:	:	

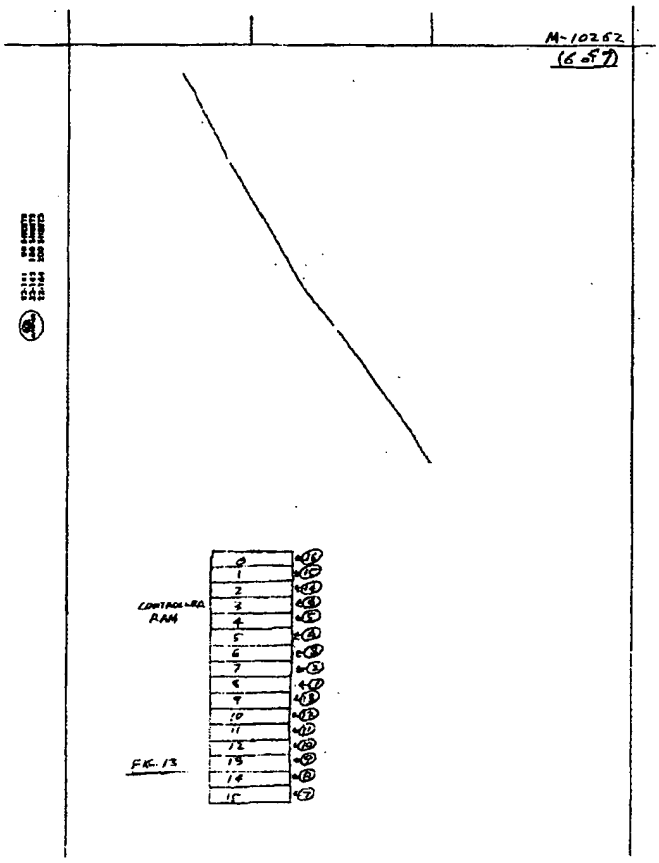


FIG. 13

M-10262
(7 of 7)

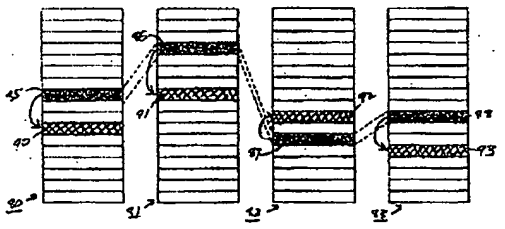


FIG. 15

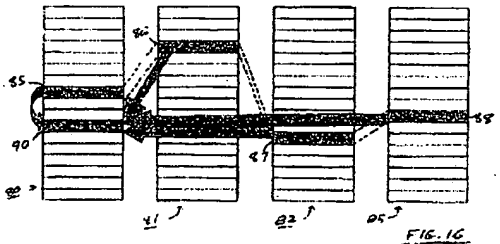


FIG. 16

【国際公開パンフレット（コレクトバージョン）】

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

CORRECTED VERSION

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
25 July 2002 (25.07.2002)

PCT

(10) International Publication Number
WO 02/058074 A2

(51) International Patent Classification⁷: G11C 16/00

(21) International Application Number: PCT/US02/00366

(22) International Filing Date: 7 January 2002 (07.01.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/766,436 19 January 2001 (19.01.2001) US

(71) Applicant: SANDISK CORPORATION [US/US]; 140 Caspian Court, Sunnyvale, CA 94089 (US).

(72) Inventor: CONLEY, Kevin, M.; 9063 Alvarado Court, San Jose, CA 95120 (US).

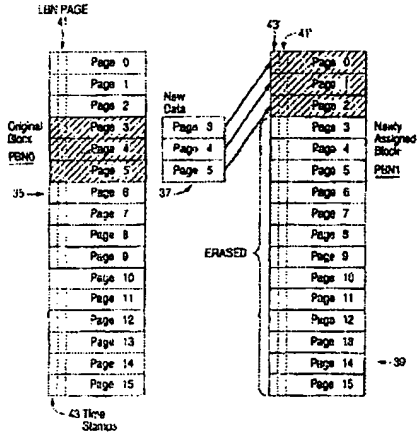
(74) Agent: PARSONS, Gerald, P.; Skjerven Morrill MacPherson LLP, Three Embarcadero Center, 28th Floor, San Francisco, CA 94111 (US).

(81) Designated States (national): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States (regional): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent

[Continued on next page]

(54) Title: PARTIAL BLOCK DATA PROGRAMMING AND READING OPERATIONS IN A NON-VOLATILE MEMORY



(57) Abstract: Data in less than all of the pages of a non-volatile memory block are updated by programming the new data in unused pages of either the same or another block. In order to prevent having to copy unchanged pages of data into the new block, or to program flags into superseded pages of data, the pages of new data are identified by the same logical address as the pages of data which they supersede and a time stamp is added to note when each page was written. When reading the data, the most recent pages of data are used and the older superseded pages of data are ignored. This technique is also applied to metablocks that include one block from each of several different units of a memory array, by directing all page updates to a single unused block in one of the units.

WO 02/058074 A2

WO 02/058074 A2

(BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report*

(48) **Date of publication of this corrected version:**
3 October 2002

(15) **Information about Correction:**
see PCT Gazette No. 40/2002 of 3 October 2002, Section II

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

PARTIAL BLOCK DATA PROGRAMMING AND READING OPERATIONS IN A NON-VOLATILE MEMORY

BACKGROUND OF THE INVENTION

This invention pertains to the field of semiconductor non-volatile data storage system architectures and their methods of operation, and has application to data storage systems based on flash electrically erasable and programmable read-only memories (EEPROMs).

A common application of flash EEPROM devices is as a mass data storage subsystem for electronic devices. Such subsystems are commonly implemented as either removable memory cards that can be inserted into multiple host systems or as non-removable embedded storage within the host system. In both implementations, the subsystem includes one or more flash devices and often a subsystem controller.

Flash EEPROM devices are composed of one or more arrays of transistor cells, each cell capable of non-volatile storage of one or more bits of data. Thus flash memory does not require power to retain the data programmed therein. Once programmed however, a cell must be erased before it can be reprogrammed with a new data value. These arrays of cells are partitioned into groups to provide for efficient implementation of read, program and erase functions. A typical flash memory architecture for mass storage arranges large groups of cells into erasable blocks, wherein a block contains the smallest number of cells (unit of erase) that are erasable at one time.

In one commercial form, each block contains enough cells to store one sector of user data plus some overhead data related to the user data and/or to the block in which it is stored. The amount of user data included in a sector is the standard 512 bytes in one class of such memory systems but can be of some other size. Because the isolation of individual blocks of cells from one another that is required to make them individually erasable takes space on the integrated circuit chip, another class of flash memories makes the blocks significantly larger so there is less space required for such isolation. But since it is also desired to handle user data in much smaller

sectors, each large block is often further partitioned into individually addressable pages that are the basic unit for reading and programming user data (unit of programming and/or reading). Each page usually stores one sector of user data, but a page may store a partial sector or multiple sectors. A "sector" is used herein to refer to an amount of user data that is transferred to and from the host as a unit.

The subsystem controller in a large block system performs a number of functions including the translation between logical addresses (LBAs) received by the memory sub-system from a host, and physical block numbers (PBNs) and page addresses within the memory cell array. This translation often involves use of intermediate terms for a logical block number (LBN) and logical page. The controller also manages the low level flash circuit operation through a series of commands that it issues to the flash memory devices via an interface bus. Another function the controller performs is to maintain the integrity of data stored to the subsystem through various means, such as by using an error correction code (ECC).

In an ideal case, the data in all the pages of a block are usually updated together by writing the updated data to the pages within an unassigned, erased block, and a logical-to-physical block number table is updated with the new address. The original block is then available to be erased. However, it is more typical that the data stored in a number of pages less than all of the pages within a given block must be updated. The data stored in the remaining pages of the given block remains unchanged. The probability of this occurring is higher in systems where the number of sectors of data stored per block is higher. One technique now used to accomplish such a partial block update is to write the data of the pages to be updated into a corresponding number of the pages of an unused erased block and then copy the unchanged pages from the original block into pages of the new block. The original block may then be erased and added to an inventory of unused blocks in which data may later be programmed. Another technique similarly writes the updated pages to a new block but eliminates the need to copy the other pages of data into the new block by changing the flags of the pages in the original block which are being updated to indicate they contain obsolete data. Then when the data are read, the updated data read from pages of the new block are combined with the unchanged data read from pages of the original block that are not flagged as obsolete.

SUMMARY OF THE INVENTION

According to one principal aspect of the present invention, briefly and generally, both the copying of unchanged data from the original to the new blocks and the need to update flags within the original block are avoided when the data of fewer than all of the pages within a block are being updated. This is accomplished by maintaining both the superceded data pages and the updated pages of data with a common logical address. The original and updated pages of data are then distinguished by the relative order in which they were programmed. During reading, the most recent data stored in the pages having the same logical address are combined with the unchanged pages of data while data in the original versions of the updated pages are ignored. The updated data can be written to either pages within a different block than the original data, or to available unused pages within the same block. In one specific implementation, a form of time stamp is stored with each page of data that allows determining the relative order that pages with the same logical address were written. In another specific implementation, in a system where pages are programmed in a particular order within the blocks, a form of time stamp is stored with each block of data, and the most recent copy of a page within a block is established by its physical location within the block.

These techniques avoid both the necessity for copying unchanged data from the original to new block and the need to change a flag or other data in the pages of the original block whose data have been updated. By not having to change a flag or other data in the superceded pages, a potential of disturbing the previously written data in adjacent pages of that same block that can occur from such a writing operation is eliminated. Also, a performance penalty of the additional program operation is avoided.

A further operational feature, which may be used in conjunction with the above summarized techniques, keeps track of the logical offset of individual pages of data within the individual memory cell blocks, so that the updated data need not be stored with the same physical page offset as the superceded data. This allows more efficient use of the pages of new blocks, and even allows the updated data to be stored in any erased pages of the same block as the superceded data.

Another principal aspect of the present invention groups together two or more blocks positioned in separate units of the memory array (also termed "sub-arrays") for

programming and reacing together as part of a single operation. Such a multiple block group is referenced herein as a "metablock." Its component blocks may be either all located on a single memory integrated circuit chip, or, in systems using more than one such chip, located on two or more different chips. When data in fewer than all of the pages of one of these blocks is updated, the use of another block in that same unit is normally required. Indeed, the techniques described above, or others, may be employed separately with each block of the metablock. Therefore, when data within pages of more than one block of the metablock are updated, pages within more than one additional block are required to be used. If there are four blocks of four different memory units that form the metablock, for example, there is some probability that up to an additional four blocks, one in each of the units, will be used to store updated pages of the original blocks. One update block is potentially required in each unit for each block of the original metablock. In addition, according to the present invention, updated data from pages of more than one of the blocks in the metablock can be stored in pages of a common block in only one of the units. This significantly reduces the number of unused erased blocks that are needed to store updated data, thereby making more efficient use of the available memory cell blocks to store data. This technique is particularly useful when the memory system frequently updates single pages from a metablock.

Additional aspects, features and advantages of the present invention are included in the following description of exemplary embodiments, which description should be read in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram of a typical prior art flash EEPROM memory array with memory control logic, data and address registers;

Figure 2 illustrates an architecture utilizing memories of Figure 1 with a system controller;

Figure 3 is a timing diagram showing a typical copy operation of the memory system of Figure 2;

Figure 4 illustrates an existing process of updating data in less than all of the pages of a multi-paged block;

Figures 5A and 5B are tables of corresponding logical and physical block addresses for each of the original and new blocks of Figure 4, respectively;

Figure 6 illustrates another existing process of updating data in less than all of the pages of a multi-paged block;

Figures 7A and 7B are tables of corresponding logical and physical page addresses for the original and new blocks of Figure 6, respectively;

Figure 8 illustrates an example of an improved process of updating data in less than all of the pages of a multi-paged block;

Figure 9 is a table of corresponding logical and physical page numbers for the new block of Figure 8;

Figure 10 provides an example of a layout of the data in a page shown in Figure 8;

Figure 11 illustrates a further development of the example of Figure 8;

Figure 12 is a table of corresponding logical and physical page numbers for the new block of Figure 11;

Figure 13 illustrates one way to read the updated data in the blocks of Figure 11;

Figure 14 is a flow diagram of a process of programming data into a memory system organized as illustrated in Figures 8 and 9; Figure 15 illustrates an existing multi-unit memory with blocks from the individual units being linked together into a metablock and

Figure 16 illustrates an improved method of updating data of a metablock in the multi-unit memory of Figure 12 when the amount of updated data is much less that the data storage capacity of the metablock.

DESCRIPTION OF EXISTING LARGE BLOCK MANAGEMENT TECHNIQUES

Figure 1 shows a typical flash memory device internal architecture. The primary features include an input/output (I/O) bus 411 and control signals 412 to interface to an external controller, a memory control circuit 450 to control internal memory operations with registers for command, address and status signals. One or more arrays 400 of flash EEPROM cells are included, each array having its own row decoder (XDEC) 401 and column decoder (YDEC) 402, a group of sense amplifiers

and program control circuitry (SA/PROG) 454 and a data register 404. Presently, the memory cells usually include one or more conductive floating gates as storage elements but other long term electron charge storage elements may be used instead. The memory cell array may be operated with two levels of charge defined for each storage element to therefore store one bit of data with each element. Alternatively, more than two storage states may be defined for each storage element, in which case more than one bit of data is stored in each element.

If desired, a plurality of arrays 400, together with related X decoders, Y decoders, program/verified circuitry, data registers, and the like are provided, for example as taught by U.S. Patent 5,890,192, issued March 30, 1999, and assigned to Sandisk Corporation, the assignee of this application, which is hereby incorporated by this reference. Related memory system features are described in co-pending patent application serial no. 09/505,555, filed February 17, 2000 by Kevin Conley et al., which application is expressly incorporated herein by this reference.

The external interface I/O bus 411 and control signals 412 can include the following:

CS - Chip Select.	Used to activate flash memory interface.
RS - Read Strobe.	Used to indicate the I/O bus is being used to transfer data from the memory array.
WS - Write Strobe.	Used to indicate the I/O bus is being used to transfer data to the memory array.
AS - Address Strobe.	Indicates that the I/O bus is being used to transfer address information.
AD[7:0] - Address/Data Bus	This I/O bus is used to transfer data between controller and the flash memory command, address and data registers of the memory control 450.

This interface is given only as an example as other signal configurations can be used to give the same functionality. Figure 1 shows only one flash memory array 400 with its related components, but a multiplicity of such arrays can exist on a single flash memory chip that share a common interface and memory control circuitry but have separate XDEC, YDEC, SA/PROG and DATA REG circuitry in order to allow parallel read and program operations.

Data is transferred from the memory array through the data register 404 to an external controller via the data registers' coupling to the I/O bus AD[7:0] 411. The data register 404 is also coupled the sense amplifier/programming circuit 454. The number of elements of the data register coupled to each sense amplifier/programming circuit element may depend on the number of bits stored in each storage element of the memory cells, flash EEPROM cells each containing one or more floating gates as the storage elements. Each storage element may store a plurality of bits, such as 2 or 4, if the memory cells are operated in a multi-state mode. Alternatively, the memory cells may be operated in a binary mode to store one bit of data per storage element.

The row decoder 401 decodes row addresses for the array 400 in order to select the physical page to be accessed. The row decoder 401 receives row addresses via internal row address lines 419 from the memory control logic 450. A column decoder 402 receives column addresses via internal column address lines 429 from the memory control logic 450.

Figure 2 shows an architecture of a typical non-volatile data storage system, in this case employing flash memory cells as the storage media. In one form, this system is encapsulated within a removable card having an electrical connector extending along one side to provide the host interface when inserted into a receptacle of a host. Alternatively, the system of Figure 2 may be embedded into a host system in the form of a permanently installed embedded circuit or otherwise. The system utilizes a single controller 301 that performs high level host and memory control functions. The flash memory media is composed of one or more flash memory devices, each such device often formed on its own integrated circuit chip. The system controller and the flash memory are connected by a bus 302 that allows the controller 301 to load command, address, and transfer data to and from the flash memory array. The controller 301 interfaces with a host system (not shown) with which user data is transferred to and from the flash memory array. In the case where the system of Figure 2 is included in a card, the host interface includes a mating plug and socket assembly (not shown) on the card and host equipment.

The controller 301 receives a command from the host to read or write one or more sectors of user data starting at a particular logical address. This address may or may not align with a boundary of a physical block of memory cells.

In some prior art systems having large capacity memory cell blocks that are divided into multiple pages, as discussed above, the data from a block that is not being updated needs to be copied from the original block to a new block that also contains the new, updated data being written by the host. This technique is illustrated in Figure 4, wherein two of a large number of blocks of memory are included. One block 11 (PBN0) is illustrated to be divided into 8 pages for storing one sector of user data in each of its pages. Overhead data fields contained within each page include a field 13 containing the LBN of the block 11. The order of the logical pages within a logical block is fixed with respect to the corresponding physical pages within a physical block. A second similarly configured block 15 (PBN1) is selected from an inventory of unused, erased blocks. Data within pages 3-5 of the original block 11 are being updated by three pages of new data 17. The new data is written into the corresponding pages 3-5 of the new block 15, and user data from pages 0-2, 6 and 7 of the block 11 are copied into corresponding pages of the new block 15. All pages of the new block 15 are preferably programmed in a single sequence of programming operations. After the block 15 is programmed, the original block 11 can be erased and placed in inventory for later use. The copying of data between the blocks 11 and 15, which involves reading the data from one or more pages in the original block and subsequently programming the same data to pages in a newly assigned block, greatly reduces the write performance and usable lifetime of the storage system.

With reference to Figures 5A and 5B, partial tables show mapping of the logical blocks into the original and new physical blocks 11 and 15 before (Figure 5A) and after (Figure 5B) the updating of data described with respect to Figure 4. Before the data update, the original block 11, in this example, stores pages 0-7 of LBN0 into corresponding pages 0-7 of PBN0. After the data update, the new block 15 stores pages 0-7 of LBN0 in corresponding pages 0-7 of PBN1. Receipt of a request to read data from LBN0 is then directed to the physical block 15 instead of the physical block 11. In a typical controller operation, a table in the form of that shown in Figures 5A and 5B is built from the LBN field 13 read from a physical page and knowledge of the PBN that is addressed when reading the data field 13. The table is usually stored in a volatile memory of the controller for ease of access, although only a portion of a complete table for the entire system is typically stored at any one time. A portion of

the table is usually formed immediately in advance of a read or programming operation that involves the blocks included in the table portion.

In other prior art systems, flags are recorded with the user data in pages and are used to indicate that pages of data in the original block that are being superceded by the newly written data are invalid. Only the new data is written to the newly assigned block. Thus the data in pages of the block not involved in the write operation but contained in the same physical block as the superceded data need not be copied into the new block. This operation is illustrated in Figure 6, where pages 3-5 of data within an original block 21 (PBN0) are again being updated. Updated pages 3-5 of data 23 are written into corresponding pages of a new block 25. As part of the same operation, an old/new flag 27 is written in each of the pages 3-5 to indicate the data of those pages is old, while the flag 27 for the remaining pages 0-2, 6 and 7 remains set at "new". Similarly, the new PBN1 is written into another overhead data field of each of the pages 3-5 in the block 21 to indicate where the updated data are located. The LBN and page are stored in a field 31 within each of the physical pages.

Figures 7A and 7B are tables of the correspondence between the data LBN/page and the PBN/page before (Figure 7A) and after (Figure 7B) the data update is complete. The unchanged pages 0-2, 6 and 7 of the LBN remain mapped into PBN0 while the updated pages 3-5 are shown to reside in PBN1. The table of Figure 7B is built by the memory controller by reading the overhead data fields 27, 29 and 31 of the pages within the block PBN0 after the data update. Since the flag 27 is set to "old" in each of pages 3-5 of the original block PBN0, that block will no longer appear in the table for those pages. Rather, the new block number PBN1 appears instead, having been read from the overhead fields 29' of the updated pages. When data are being read from LBN0, the user data stored in the pages listed in the right column of Figure 7B are read and then assembled in the order shown for transfer to the host.

Various flags are typically located in the same physical page as the other associated overhead data, such as the LBN and an ECC. Thus, to program the old/new flags 27, and others, in pages where the data has been superceded requires that a page support multiple programming cycles. That is, the memory array must have the capability that its pages can be programmed in at least at least two stages between erasures. Furthermore, the block must support the ability to program a page

when other pages in the block with higher offsets or addresses have been already programmed. A limitation of some flash memories however prevents the usage of such flags by specifying that the pages in a block can only be programmed in a physically sequential manner. Furthermore, the pages support a finite number of program cycles and in some cases additional programming of programmed pages is not permitted.

What is needed is a mechanism by which data that partially supercedes data stored in an existing block can be written without either copying unchanged data from the existing block or programming flags to pages that have been previously programmed.

DESCRIPTION OF EXEMPLARY EMBODIMENTS OF THE INVENTION

There are many different types of flash EEPROM, each of which presents its own limitations that must be worked around to operate a high performance memory system formed on a small amount of integrated circuit area. Some do not provide for writing any data into a page that has already been programmed, so updating flags in a page that contains superceded data, as described above, is not possible. Others allow such flags to be written but doing so in pages whose data is being superceded can disturb data in other pages of the same block that remain current.

An example memory system where this has been found to be a problem is a NAND type, where a column of memory cells is formed as a series circuit string between a bit line and a common potential. Each word line extends across a row of memory cells formed of one cell in each such string. Such a memory is particularly susceptible to such memory state disturbs when being operated in a multi-state mode to store more than one bit of data in each such cell. Such operation divides an available window of a memory cell transistor threshold voltage range into narrow non-overlapping voltage level ranges, each range becoming narrower as the number of levels, and thus the number of bits being stored in each cell, are increased. For example, if four threshold ranges are used, two bits of data are stored in each cell's storage element. And since each of the four threshold voltage ranges is necessarily small, the chance of the state of a cell being disturbed by programming other cells in the same block is increased with multi-state operation. In this case, the writing of the

old/new or other flags, as described with respect to Figures 6, 7A and 7B, cannot be tolerated.

A common feature of each of the existing memory management techniques described above with respect to Figures 4-7B is that a logical block number (LBN) and page offset is mapped within the system to at most two physical block numbers (PBNs). One block is the original block and the other contains the updated page data. Data are written to the page location in the block corresponding to the low order bits of its logical address (LBA). This mapping is typical in various types of memory systems. In the techniques described below, pages containing updated data are also assigned the same LBN and page offsets as the pages whose data has been superceded. But rather than tagging the pages containing original data as being superceded, the memory controller distinguishes the pages containing the superceded data from those containing the new, updated version either (1) by keeping track of the order in which the pages having the same logical addresses were written, such as by use of a counter, and/or (2) from the physical page addresses wherein, when pages are written in order within blocks from the lowest page address to the highest, the higher physical address contains the most recent copy of the data. When the data is accessed for reading, therefore, those in the most current pages are used in cases where there are pages containing superceded data that have the same logical addresses, while the superceded data are ignored.

A first specific implementation of this technique is described with respect to Figures 8 and 9. The situation is the same in this example as that in the prior art techniques described with respect to Figures 4-7B, namely the partial re-write of data within a block 35, although each block is now shown to contain 16 pages. New data 37 for each of the pages 3-5 of the block 35 (PBN 35) is written into three pages of a new block 39 (PBN1) that has previously been erased, similar to that described previously. A LBN and page offset overhead data field 41 written into the pages of PBN1 that contain the updated data is the same as that in the pages of the superceded data in the initial block PBN0. The table of Figure 9, formed from the data within the fields 41 and 41', shows this. The logical LBN and page offsets, in the first column, are mapped into both the first physical block (PBN0), in the second column, and, for the pages that have been updated, also into the second physical block (PBN1) in the third column. The LBN and logical page offsets 41' written into each of the three

pages of updated data within the new block PBN1 are the same as those 41 written into each of a corresponding logical page of the original block PBN0.

In order to determine which of two pages having the same LBN and page offset contains the updated data, each page contains another overhead field 43 that provides an indication of its time of programming, at least relative to the time that other pages with the same logical address are programmed. This allows the controller to determine, when reading the data from the memory, the relative ages of the pages of data that are assigned the same logical address.

There are several ways in which the field 43, which contains a form of time stamp, may be written. The most straight forward way is to record in that field, when the data of its associated page is programmed, the output of a real-time clock in the system. Later programmed pages with the same logical address then have a later time recorded in the field 43. But when such a real-time clock is not available in the system, other techniques can be used. One specific technique is to store the output of a modulo-N counter as the value of the field 43. The range of the counter should be one more than the number of pages that are contemplated to be stored with the same logical page number. When updating the data of a particular page in the original block PBN0, for example, the controller first reads the count stored in the field 43 of the page whose data are being updated, increments the count by some amount, such as one, and then writes that incremented count in the new block PBN1 as the field 43'. The counter, upon reaching a count of N+1, rolls over to 0. Since the number of blocks with the same LBN is less than N, there is always a point of discontinuity in the values of stored counts. It is easy then to handle the rollover with normalized to the point of discontinuity.

The controller, when called upon to read the data, easily distinguishes between the new and superseded pages' data by comparing the counts in the fields 43 and 43' of pages having the same LBA and page offset. In response to a need to read the most recent version of a data file, data from the identified new pages are then assembled, along with original pages that have not been updated, into the most recent version of the data file.

It will be noted that, in the example of Figure 8, the new data pages 37 are stored in the first three pages 0-2 of the new block PBN1, rather than in the same pages 3-5 which they replace in the original block PBN0. By keeping track of the

individual logical page numbers, the updated data need not necessarily be stored in the same page offset of the new block as that of the old block where superceded data is contained. Page(s) of updated data can also be written to erased pages of the same block as the page of data being superceded.

As a result, there is no constraint presented by the techniques being described that limit which physical page new data can be written into. But the memory system in which these techniques are implemented may present some constraints. For example, one NAND system requires that the pages within the blocks be programmed in sequential order. That means that programming of the middle pages 3-5, as done in the new block 25 (Figure 6), wastes the pages 0-2, which cannot later be programmed. By storing the new data 37 in the first available pages of the new block 39 (Figure 8) in such a restrictive system, the remaining pages 3-7 are available for later use to store other data. Indeed, if the block 39 had other data stored in its pages 0-4 at the time the three pages of new data 37 were being stored, the new data could be stored in the remaining unused pages 5-7. This makes maximum use of the available storage capacity for such a system.

An example of the structure of data stored in an individual page of the blocks of Figure 8 is shown in Figure 10. The largest part is user data 45. An error correction code (ECC) 47 calculated from the user data is also stored in the page. Overhead data 49, including the LBN and page tag 41 (logical page offset), the time stamp 43 and an ECC 51 calculated from the overhead data are also stored in the page. By having an ECC 50 covering the overhead data that is separate from the user data ECC 47, the overhead 49 may be read separately from the user data and evaluated as valid without the need to transfer all of the data stored in the page. Alternatively, however, where the separate reading of the overhead data 49 is not a frequent event, all of the data in the page may be covered by a single ECC in order to reduce the total number of bits of ECC in a page.

A second specific implementation of the inventive technique can also be described with respect to Figure 8. In this example, the time stamp is used only to determine the relative age of the data stored in blocks, while the most recent pages among those that carry the same LBN and page number are determined by their relative physical locations. The time stamp 43 then does not need to be stored as part of each page. Rather, a single time stamp can be recorded for each block, either as

part of the block or elsewhere within the non-volatile memory, and is updated each time a page of data is written into the block. Data is then read from pages in an order of descending physical address, starting from the last page of the most recently updated block containing data pages having the same LBN.

In Figure 8, for example, the pages are first read in the new block PBN1 from the last (page 15) to the first (page 0), followed by reading the pages of the original block PBN0 in the same reverse order. Once logical pages 3, 4 and 5 have been read from the new block PBN1, the superceded data in those pages of the original block PBN0 that are identified by the same logical page numbers can be skipped during the reading process. Specifically, physical pages 3, 4 and 5 of the old block PBN0 are skipped during reading, in this example, once the controller determines that their LBN/pages 41 are the same as those of the pages already read from the new block PBN1. This process can increase the speed of reading and reduce the number of overhead bits 49 that need to be stored for each page. Further, when this reverse page reading technique is employed, the table of Figure 9 used by the controller during a reading operation can be simplified into the form of Figures 5A and 5B. Only an identity of those physical blocks containing data of a common logical block and the relative times that the physical blocks were programmed need to be known in order to carry out this efficient reading process.

Figure 11 illustrates an extension of the example of Figure 8 by including a second update to the data originally written in the block PBN0. New data 51 for logical pages 5, 6, 7 and 8 is written to the respective physical pages 3, 4, 5 and 6 of the new block PBN1, along with their LBN and page number. Note, in this example, that the data of logical page 5 is being updated for the second time. During a reading operation that begins from the last page of the new block PBN1, the most recently written logical pages 8, 7, 6 and 5 of the data of interest are first read in that order. Thereafter, it will be noted that the LBN/page overhead field in physical page 2 of PBN1 is the same as that *read* from the physical page 3, so the user data of page 2 is not read. The physical pages 1 and 0 are then read. Next, the pages of the original block PBN0 are read, beginning with physical page 15. After reading physical pages 15-9, the controller will note that the LBN/page fields of each of pages 8-3 match those of pages whose data has already been read, so the old data need not be read

from those pages. The efficiency of the reading process is thus improved. Finally, the original data of physical pages 2-0 are read since that data was not updated.

It will be noted that this example of reading pages in a reverse order efficiently sorts out the new data pages from the superceded data pages because data are written in physical page locations of an erased block in order from page 0 on. This technique is not limited to use with a memory system having such a specific programming constraint, however. So long as the order in which pages are programmed within a given block is known, the data from those pages may be read in the reverse order from which they were written. What is desired is that the most recently programmed pages having a common LBN with others that were earlier programmed be read first, and these are the most recently programmed pages. The most recent versions of updated pages are read first so that the superceded versions may easily be identified thereafter.

A table showing the correspondence between the logical data and physical page addresses for the example of Figure 11 is given in Figure 12. Although there have been two data updates, both are represented by the single column for the second block PBN1. The physical page noted in PBN1 for the logical page 5 is simply changed upon the second update to that page occurring. If the updating involves a third block, then another column is added for that other block. The table of Figure 12, constructed by reading the overhead data from each of the pages in blocks to which data of a common LBN has been written, can be used by the first implementation when the reverse page reading technique is not used. When the reverse page reading technique described above is used, the table of Figure 12 need be built only to identify a correspondence between an LBN and all PBNs containing data of that LBN.

An efficient way to organize pages of data being read from a physical block, where one or more of the pages has been updated, is illustrated by Figure 13. Enough space is provided in a volatile memory of the controller to buffer at least several pages of data at a time, and preferably a full block of data. That is what is shown in Figure 13. Sixteen pages of data, equal to the amount stored in a non-volatile memory block, are stored in the controller memory. Since the pages are most commonly read out of order, each page of data is stored in its proper position with respect to the other pages. For example, in the reverse page read operation of Figure 11, logical page 8 if the first to be read, so it is stored in position 8 of the controller memory, as indicated by the

"1" in a circle. The next is logical page 7, and so forth, until all pages of data desired by the host are read and stored in the controller memory. The entire set of page data is then transferred to the host without having to manipulate the order of the data in the buffer memory. The pages of data have already be organized by writing them to the proper location in the controller memory.

A method of programming a non-volatile memory system that utilizes the techniques described with respect to Figures 8 and 9 is illustrated in the flow chart of Figure 14. Data for pages of an existing file to be updated are received from a host system, as indicated by the block 52. It is first determined by a step 53 whether the number of pages of updated data to be stored is equal to or greater than the storage capacity of a block of the system, 16 pages being shown as the block capacity, for simplicity, in the above described example. If so, one or more unused, erased blocks are addressed, in a step 55, and the new data pages are written to the addressed block(s), in a step 57. Typically, the updating of one block or more of data will result in one or more blocks storing the data that have been superceded by the new data. If so, as indicated by a step 59, those blocks with superceded data are identified for erasure. For the purpose of increasing performance, it is preferable that erase operations occur in the background, or when host requested programming or reading operations are not taking place. After being erased, the blocks are returned to the inventory of unused, erased blocks for further use. Alternatively, erasure of the blocks can be deferred until they are needed for programming operations.

If, on the other hand, in the step 53, it is determined that there are fewer pages of new data than will utilize the full storage capacity of a block, a next step 61 determines whether there are enough unused pages in a block having some pages programmed with other data. If so, such a block is addressed, in a step 63. If not, a totally unused, erased block is addressed, in a step 65. In either case, in a step 67, the new data are programmed into unused pages of the addressed block. As part of this programming process, the LBN and page offset is written into the fields 41, and the time stamp into the fields 43 of each of the pages (Figure 8) of the updated data, in the manner described above.

A desirable feature of the programming process is to make available for future programming any blocks that store only superceded data. So the question is asked, in a step 69, whether the data updating process has resulted in an entire block remaining

with only superceded data. If so, such a block is queued for erasure, in a step 71, and the process is then completed. If not, the step 71 is omitted and the data update is finished.

METABLOCK OPERATION

In order to improve performance by reducing programming time, a goal is to program as many cells in parallel as can reasonably be done without incurring other penalties. One implementation divides the memory array into largely independent sub-arrays or units, such as multiple units 80-83 of Figure 15, each unit in turn being divided into a large number of blocks, as shown. Pages of data are then programmed at the same time into more than one of the units. Another configuration further combines one or more of these units from multiple memory chips. These multiple chips may be connected to a single bus (as shown in Figure 2) or multiple independent busses for higher data throughput. An extension of this is to link blocks from different units for programming, reading and erasing together, an example being shown in Figure 15. Blocks 85-88 from respective ones of the units 80-83 can be operated together as a metablock, for example. As with the memory embodiments described above, each block, the smallest erasable group of the memory array, is typically divided into multiple pages, a page containing the smallest number of cells that are programmable together within the block. Therefore, a programming operation of the metablock shown in Figure 15 will usually include the simultaneously programming of data into at least one page of each of the blocks 85-88 forming the metablock, which is repeated until the metablock is full or the incoming data has all been programmed. Other metablocks are formed of different blocks from the array units, one block from each unit.

In the course of operating such a memory, as with others, pages of data less than an entire block often need to be updated. This can be done for individual blocks of a metablock in the same manner as described above with respect to either of Figures 4 or 6, but preferably by use of the improved technique described with respect to Figure 8. When any of these three techniques are used to update data of one block of the metablock, an additional block of memory within the same unit is also used. Further, a data update may require writing new data for one or more pages of two or more of the blocks of a metablock. This can then require use of up to four additional

blocks 90-93, one in each of the four units, to update a data file stored in the metablock, even though the data in only a few pages is being updated.

In order to reduce the number of blocks required for such partial block updates, according to another aspect of the present invention, updates to pages of data within any of the blocks of the illustrated metablock are made, as illustrated by Figure 16, to a single additional block 90 in the memory unit 80, so long as unused pages in the block 80 remain. If, for example, data in three pages of the block 86 and two pages of the block 88 are being updated at one time, all five pages of the new data are written into the block 90. This can save the use of one block of memory, thereby to effectively increase the number of available erased blocks by one block. This helps avoid, or at least postpone, the time when an inventory of erased blocks becomes exhausted. If one or more pages from each of the four blocks 85-88 are being updated, all of the new data pages are programmed in the single block 90, thereby avoiding tying up an additional three blocks of memory to make the update. If the number of pages of new data exceed the capacity of an unused block, pages that the block 90 cannot accept are written to another unused block which may be in the same unit 80 or one of the other units 81-83.

Although the invention has been described with respect to various exemplary embodiments, it will be understood that the invention is entitled to protection within the full scope of the appended claims.

IT IS CLAIMED:

1. A method of simultaneously storing original and replacement data in a non-volatile memory system, comprising:

identifying the original and replacement data by the same logical address, and

distinguishing the replacement data from the original data by keeping track of the relative times that the original and replacement data have been programmed into the memory.

2. A method of storing and retrieving original and replacement data in a non-volatile memory system, comprising:

identifying units of the original and the replacement data by the same logical address,

reading units of data in an inverse order from an order in which they were programmed into the memory, and

distinguishing units of replacement data from units of original data having the same logical address by the order in which they are read.

3. In a non-volatile memory system having a plurality of blocks of memory storage elements that are individually organized into a plurality of pages of memory storage elements, a method of substituting new data for superceded data within at least one page of one of the plurality of blocks while data in at least another page of said one block is not replaced, comprising:

programming the new data into at least one page of said one or another of the plurality of blocks,

identifying the at least one page of superceded data and the at least one page of new data by a common logical address, and

recording a relative time of programming the new and the superceded data.

4. The method of claim 3, wherein the relative time of programming is recorded for the individual pages in which the new and superceded data are programmed, whereby the at least one page of new data is distinguishable from the at least one page of superceded data by their recorded relative times of programming.

5. The method of claim 3, wherein the relative time of programming is recorded for the individual blocks, thereby to identify an order of programming of individual blocks having data with a common logical address, and further wherein pages within the individual blocks are programmed in a designated order, whereby the new pages of data are distinguishable from superceded pages of data within an individual block by their relative positions within the block.

6. The method of claim 3, wherein the data in at least another page of said one block that is not replaced are not copied into said one or another block as part of substituting the new data for the superceded data.

7. The method of claim 3, wherein nothing is written into the at least one page of superceded data as part of substituting the new data for the superceded data.

8. The method of claim 4, wherein recording a relative time of programming the new and superceded data includes storing a value of a clock at each of the times that the new and superceded data are programmed.

9. The method of claim 4, wherein recording a relative time of programming the new and superceded data includes storing a different value of a sequence of numbers at each of the times that the new and superceded data are programmed.

10. The method of either of claims 8 or 9, wherein storing the value indicating a relative time of programming the new and superceded data includes storing the individual values within the same pages as the new and superceded data to which the values relate.

11. The method of claim 3, wherein programming the new data into at least one page of another said one or another of the plurality of blocks includes programming the new data into the first available unused pages within said one or another block in a predefined order.

12. The method of claim 3, wherein identifying the at least one page of superceded data and the at least one page of new data by a common logical address includes recording at least part of the common logical address in the individual pages as overhead data.

13. The method of claim 12, including building a table in volatile memory including multiple physical block addresses for the common logical address.

14. A method of reading data that has been updated according to claim 4, comprising:

reading pages of data from said one block and, if new data has been programmed thereinto, said another block,

identifying any multiple pages of data that have the same logical address,

utilizing the recorded relative time of programming the new and superceded data to identify the most current of any pages having the same logical address, and

assembling data in the most current of any pages having the same logical address along with pages in said at least another page of said one block that have not been updated.

15. A method of reading data that has been updated according to claim 5, comprising:

reading pages of data within said one and, if new data has been programmed thereinto, another block in a reverse order from which they were programmed, and

passing over any pages of data so read which have the same logical page address as a page whose data has already been read.

16. The method of either one of claims 14 or 15, additionally comprising oporating the individual memory storage elements with more than two storage states, thereby storing more than one bit of data in each storage element, and reading pages of data includes reading the more than two storage states from the individual memory storage elements.

17. The method of any one of claims 3-9, additionally comprising operating storage elements of the individual memory cells with more than two storage states, thereby storing more than one bit of data in each storage element.

18. The method of claim 17, wherein the storage elements include individual floating gates.

19. The method of any one of claims 3-9, wherein the non-volatile memory system is formed within an enclosed card having an electrical connector along one edge thereof that operably connects with a host system.

20. A method of operating a non-volatile memory system having an array of memory storage elements organized into at least two sub-arrays, wherein the individual sub-arrays are divided into a plurality of non-overlapping blocks of storage elements wherein a block contains the smallest group of memory storage elements that are erasable together, and the individual blocks are divided into a plurality of pages of storage elements wherein a page is the smallest group of memory storage elements that are programmable together, comprising:

linking at least one block from individual ones of said at least two sub-arrays to form a metablock wherein its component blocks are erased together as a unit, and

updating pages of original data within any of the metablock component blocks less than all the pages within the block by programming replacement data into pages within another at least one block in only a designated one of the sub-arrays regardless of which sub-array the data being updated is stored.

21. The method of claim 20, wherein storing the original and replacement data includes:

identifying the original and replacement data by the same logical address to the memory system, and

distinguishing the replacement data from the original data by keeping track of the relative times that the original and replacement data have been programmed their respective pages of the memory.

22. A non-volatile memory system, comprising:

an array of non-volatile memory storage elements organized in blocks of storage elements, wherein an individual block contains the smallest group of storage elements that is erasable, and

a programming mechanism that writes into a first block an updated version of less than all of original data stored in a second block along with an indication of the later writing of the updated version,

an address mechanism that logically addresses both the original data and the updated version with the same address, and

a reading mechanism that distinguishes the updated version from the original data at least in part by the relative time by said indication of the later writing of the updated version.

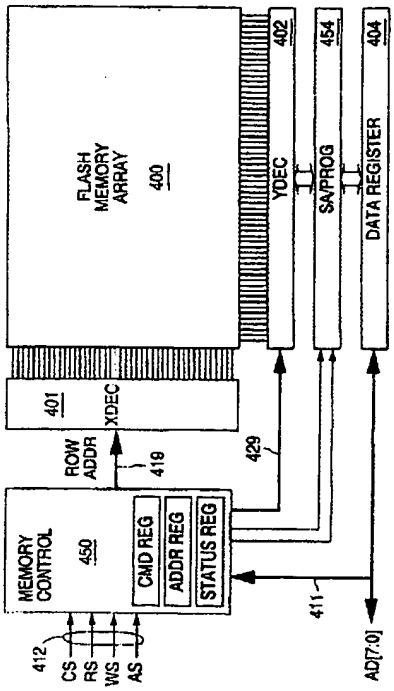


FIG. 1
(PRIOR ART)

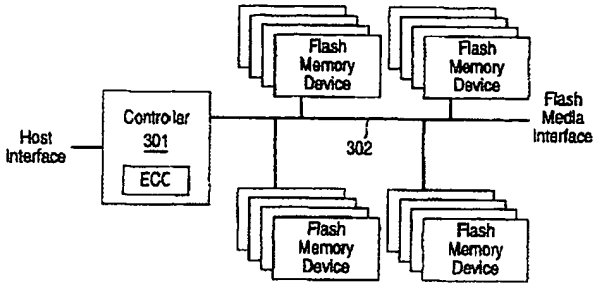


FIG. 2
(PRIOR ART)

LBN	PBN
0	0

Original Block 11

FIG. 5A

LBN	PBN
0	1

With New Block 15

FIG. 5B

FIG. 3
(PRIOR ART)

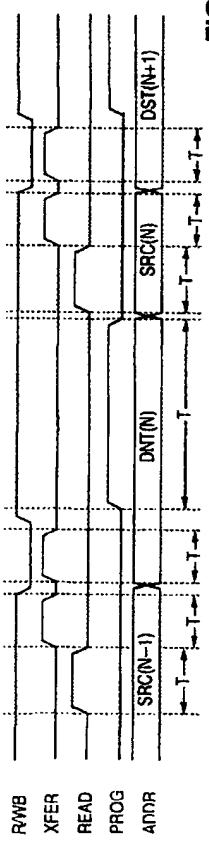
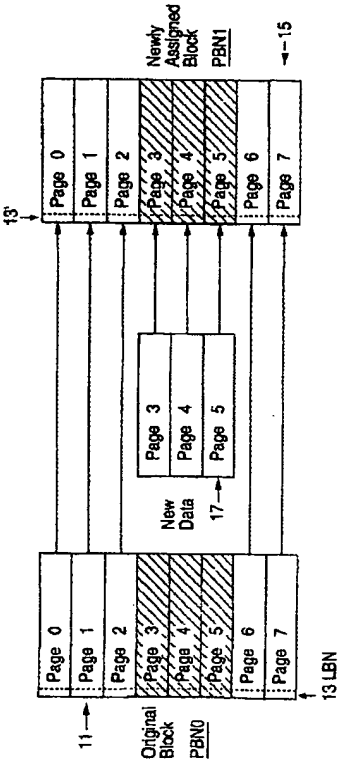


FIG. 4
(PRIOR ART)



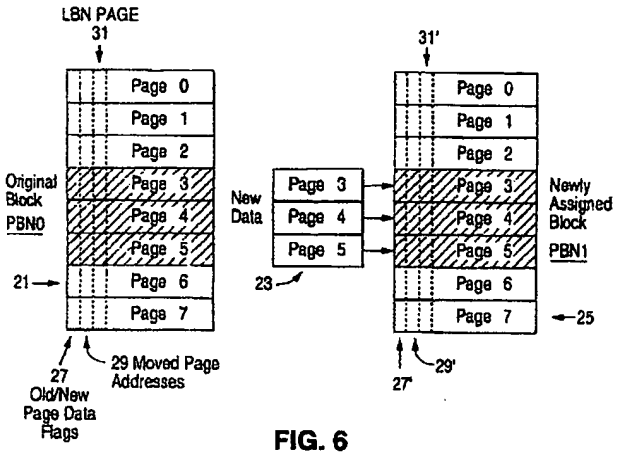


FIG. 6
(PRIOR ART)

LBN	Page	PBN	Page
0	0	0	0
0	1	0	1
0	2	0	2
0	3	0	3
0	4	0	4
0	5	0	5
0	6	0	6
0	7	0	7
:	:	:	:

Original Block

FIG. 7A

LBN	Page	PBN	Page
0	0	0	0
0	1	0	1
0	2	0	2
0	3	1	3
0	4	1	4
0	5	1	5
0	6	0	6
0	7	0	7
:	:	:	:

With New Block

FIG. 7B

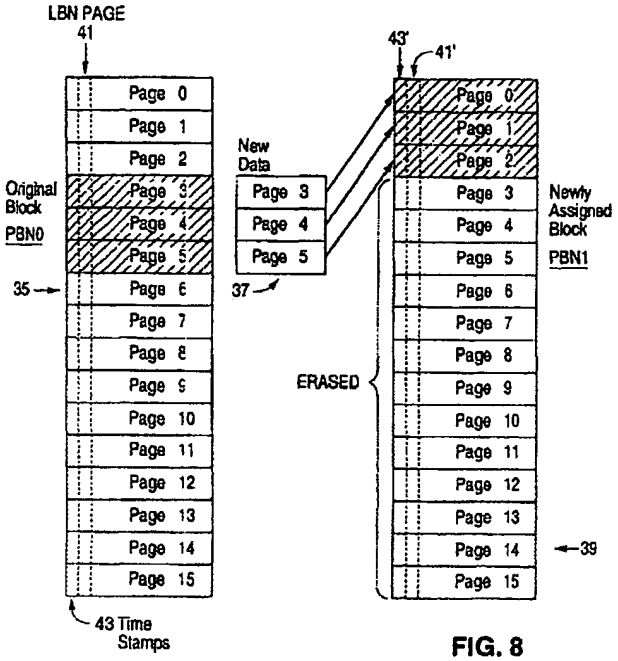


FIG. 8

LBN	Page	PBN0	Page	PBN1	Page
0	0	0	0		
0	1	0	1		
0	2	0	2		
0	3	0	3	1	0
0	4	0	4	1	1
0	5	0	5	1	2
0	6	0	6		
0	7	0	7		

FIG. 9

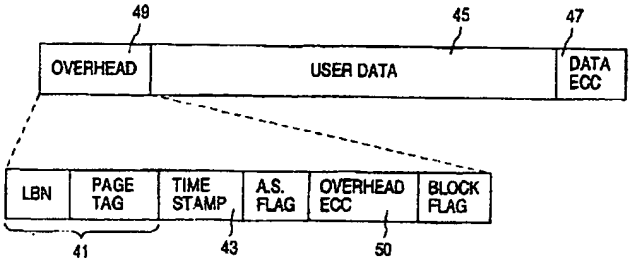


FIG. 10

LBN	Page	PBN0	Page	PBN1	Page
0	0	0	0		
0	1	0	1		
0	2	0	2		
0	3	0	3	1	0
0	4	0	4	1	1
0	5	0	5	1	3
0	6	0	6	1	4
0	7	0	7	1	5
0	8	0	8	1	6
0	9	0	9		
:	:	:	:	:	

FIG. 12

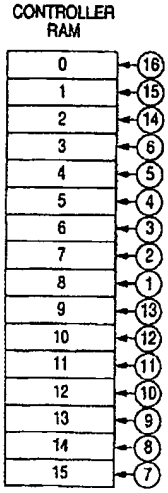
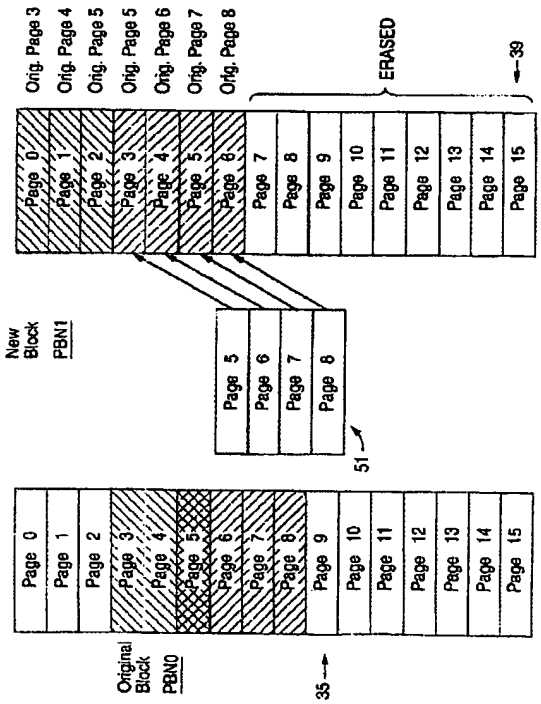


FIG. 13

JP 2004-533029 A 2004.10.28

WO 02/058074 PCT/US02/00366

FIG. 11



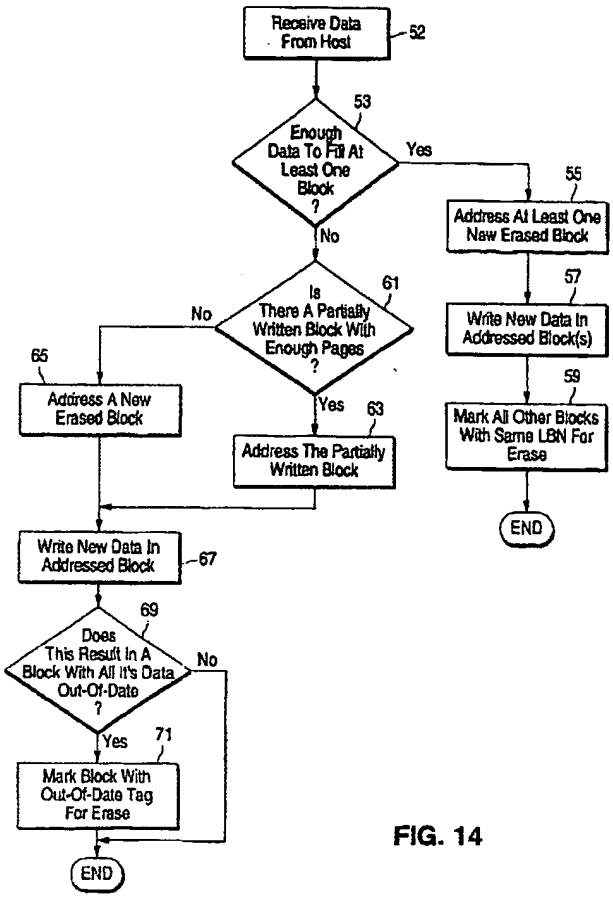

Receive Data From Host
52
Enough Data To Fill At Least One Block ?
53
Yes
No
Address At Least One New Erased Block
55
Write New Data In Addressed Block(s)
57
Mark All Other Blocks With Same LBN For Erase
59
END
Is There A Partially Written Block With Enough Pages ?
61
No
Yes
Address The Partially Written Block
63
Address A New Erased Block
65
Write New Data In Addressed Block
67
Does This Result In A Block With All It's Data Out-Of-Date ?
69
No
Yes
Mark Block With Out-Of-Date Tag For Erase
71
END

FIG. 14

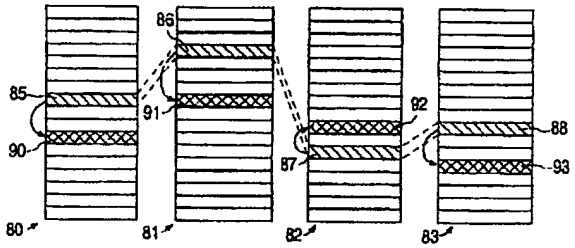


FIG. 15

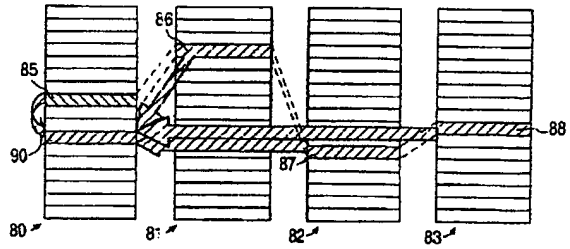


FIG. 16

【国際公開パンフレット（コレクトバージョン）】

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
25 July 2002 (25.07.2002)

PCT

(10) International Publication Number
WO 02/058074 A3

(51) International Patent Classification⁷: G06F 12/02, 3/06

(21) International Application Number: PCT/US02/00366

(22) International Filing Date: 7 January 2002 (07.01.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/765,416　19 January 2001 (19.01.2001)　US

(71) Applicant: SANDISK CORPORATION [US/US]; 140 Caspian Court, Sunnyvale, CA 94089 (US).

(72) Inventor: CONLEY, Kevin, M.; 5983 Alhambra Court, San Jose, CA 95120 (US).

(74) Agent: PARSONS, Gerald, P.; Parsons Hsue & de Runtz LLP, 655 Montgomery Street, Suite 1800, San Francisco, CA 94111 (US).

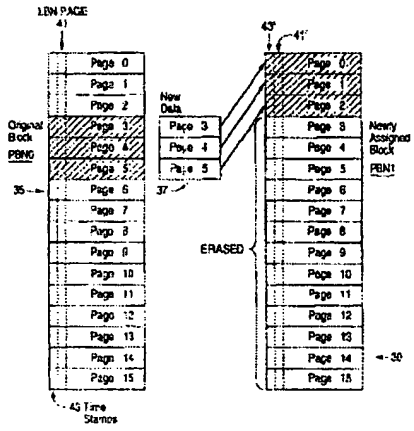
(81) Designated States (national): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States (regional): ARIPO patent: (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— with international search report

[Continued on next page]

(54) Title: PARTIAL BLOCK DATA PROGRAMMING AND READING OPERATIONS IN A NON-VOLATILE MEMORY



(57) Abstract: Data in less than all of the pages of a non-volatile memory block are updated by programming the new data in unused pages of either the same or another block. In order to prevent having to copy unchanged pages of data into the new block, or to program flags into superseded pages of data, the pages of new data are identified by the same logical address as the pages of data which they superseded and a time stamp is added to note when each page was written. When reading the data, the most recent pages of data are used and the older superseded pages of data are ignored. This technique is also applied to metablocks that include one block from each of several different units of a memory array, by directing all page updates to a single unused block in one of the units.

WO 02/058074 A3

WO 02/058074 A3

(88) Date of publication of the international search report:
10 July 2003

(15) Information about Correction:
Previous Correction:
see PCT Gazette No. 40/2002 of 3 October 2002, Section II

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

【国際調査報告】

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 02/00366

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F12/02 G06F3/06

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F G11C

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, PAJ

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X A	US 6 698 370 A (ASANO HIDEO ET AL) 28 January 1997 (1997-01-28) column 3, line 13 -column 4, line 14	1-4, 6-15,22 20,21
E	WO 02 49039 A (SANDISK CORP) 20 June 2002 (2002-06-20) page 13, paragraph 3 -page 14, paragraph 1	1-4,8,9, 12,13, 17-19,22
A	FR 2 742 893 A (SCHLUMBERGER IND SA) 27 June 1997 (1997-06-27) page 1, line 4 -page 3, line 32; figure 1	1-11, 19-22

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
31 March 2003	07/04/2003

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Nielsen, O

Form PCT/ISA/210 (second sheet) (July 1998)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 02/00366

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This International Search Report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the International Application that do not comply with the prescribed requirements to such an extent that no meaningful International Search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

see additional sheet

1. ☐ As all required additional search fees were timely paid by the applicant, this International Search Report covers all searchable claims.

2. ☒ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this International Search Report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this International Search Report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☐ No protest accompanied the payment of additional search fees.

International Application No. PCT/US 02 00366

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

This International Searching Authority found multiple (groups of) inventions in this international application, as follows:

1. Claims: 1-19 22

Method for storing replacement data in a non-volatile memory by keeping track of relative times of writing

2. Claim : 20 21

Method of updating: pages within a metablock of non-volatile memory system comprising multiple sub-arrays by writing updated data to only a single subarray.

INTERNATIONAL SEARCH REPORT

tion on patent family members

Internat Application No
PCT/US 02/00366

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
US 5598370	A	28-01-1997	JP 2856621 B2 JP 6250798 A US 5457658 A		10-02-1999 09-09-1994 10-10-1995
WO 0249039	A	20-06-2002	AU 3955102 A WO 0249039 A2		24-06-2002 20-06-2002
FR 2742893	A	27-06-1997	FR 2742893 A1		27-06-1997

Form PCT/ISA/210 (patent family annex) (July 1992)

フロントページの続き

(81)指定国 AP(GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), EA(AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), EP(AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OA(BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG), AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☑ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☑ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☑ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.